VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

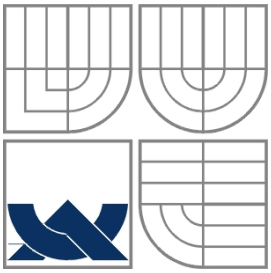
APLIKACE PRO SLEDOVÁNÍ DOPRAVY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

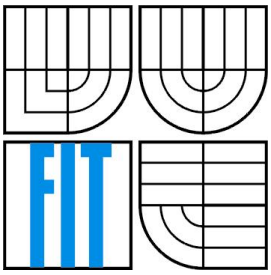
AUTOR PRÁCE
AUTHOR

FILIP TESAŘ

BRNO 2012



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

APLIKACE PRO SLEDOVÁNÍ DOPRAVY

SOFTWARE FOR TRAFFIC MONITORING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR
FILIP TESAŘ

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. MICHAL ŠPANĚL, Ph.D.

BRNO 2012

Abstrakt

Tato práce se zabývá detekcí a sledováním pohybujících se objektů ve videosekvenci. Cílem práce je návrh a implementace aplikace, která z videosekvence zachycující dopravní provoz získá statistická data o jednotlivých vozidlech, například odhadne rychlost vozidla a určí jeho jízdní pruh. Vytvořená aplikace používá metodu odečítání pozadí Mixture of Gaussians a umožňuje sledování více dopravních úseků současně nezávislými kamerami, výstupní údaje ukládá do databáze a uživateli umožňuje jejich zobrazení ve formě grafů skrze jednoduché webové rozhraní.

Abstract

This thesis deals with detection and tracking of moving objects in video sequences. The aim of this work is design and implementation of an application, which uses video sequence capturing car traffic to acquire statistical data about each vehicle, such as its speed estimate and determination of its lane. The proposed application uses background subtraction method 'Mixture of Gaussians' and enables monitoring of multiple locations at the same time using independent cameras. The output data are saved into database and users are allowed to display statistics in graphs through a simple web interface.

Klíčová slova

sledování dopravy, dohledový systém, detekce pohybu, sledování objektů, metody odečítání pozadí, Mixture of Gaussians, ASP.NET

Keywords

traffic monitoring, traffic surveillance, motion detection, object tracking, methods of background subtraction, Mixture of Gaussians, ASP.NET

Citace

Filip Tesař: Aplikace pro sledování dopravy, bakalářská práce, Brno, FIT VUT v Brně, 2012

Aplikace pro sledování dopravy

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Michala Španěla, Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Filip Tesař
16. 5. 2012

Poděkování

Chtěl bych poděkovat vedoucímu mé bakalářské práce Ing. Michalu Španělovi, Ph.D. za jeho odborné vedení, ochotu pomoci a cennou zpětnou vazbu, kterou mi poskytoval po celou dobu tvorby této práce. Dále bych chtěl poděkovat své přítelkyni, rodině a kamarádům za podporu a zázemí.

Obsah

1 Úvod

Dohledové systémy se v současné době rychle rozšiřují, ve městech narůstá počet kamer určených k dozoru pořádku v ulicích, stejně tak soukromé objekty jsou stále častěji zabezpečovány kamerovým systémem. Kamery jsou umístěny i na důležitých dopravních křižovatkách a tepnách. Není ale v silách člověka obraz všech těchto kamer neustále monitorovat, aby mohl rychle reagovat na vzniklé situace. Právě z tohoto důvodu je vysoká poptávka po automatických dohledových systémech, které by situace odehrávající se před objektivy kamer sami vyhodnocovaly a člověka upozornily pouze v případě, kdy došlo k odchýlení od běžného stavu.

Cílem této práce je navrhnout a implementovat aplikaci pro sledování dopravy, jejímž vstupem bude signál z videokamery. Aplikace bude sbírat statistická data o dopravě na sledovaném úseku komunikace, konkrétně bude počítat jednotlivá vozidla, odhadovat jejich rychlost a detekovat jízdní pruh, ve kterém se vozidla pohybují. Z těchto dat bude následně vytvářet grafy pro analýzu dopravní situace. Vytvořená aplikace bude umožňovat sledování více dopravních úseků současně nezávislými kamerami a jednou z jejích funkcí bude i zjednodušené zobrazení aktuálních dopravních stavů více úseků na jedné obrazovce. Člověk konající dohled by tedy měl být schopen z této obrazovky zjistit, že na určitém úseku došlo k problému, aniž by sledoval přímo obraz jednotlivých kamer. Aplikace bude rozdělena na dvě části – modul sbírající data a webovou aplikaci zobrazující statistiky formou grafů. Propojení mezi oběma částmi bude zajištěno relační databází.

Jedním z hlavních řešených problémů je tedy detekce pohybujících se vozidel ve videosekvenci, čemuž je věnována 2. kapitola. Je v ní vysvětlen princip metod odečítajících pozadí, pozornost bude zaměřena hlavně na metodu Mixture of Gaussians. Detekovaná vozidla je poté nutné v záběru sledovat, touto problematikou se zabývá 3. kapitola. Návrhu systému, který v rámci této práce vznikne, se věnuje 4. kapitola. V 5. kapitole je popsána implementace obou částí výsledného systému včetně krátkého popisu použitých technologií a knihoven. Výsledky práce jsou uvedeny v 6. kapitole, která také obsahuje podrobnosti o testování aplikace.

2 Detekce pohybujících se objektů ve videosekvenci

V této kapitole budou popsány některé používané metody k detekci pohybujících se objektů ve videosekvenci natočené statickou kamerou. Z hlediska počítačového vidění se jedná o netriviální problém, protože vstupním signálem detekčních algoritmů je pouze dvourozměrná (v případě šedotónového obrazu) nebo trojrozměrná (v případě barevného RGB modelu) matice čísel. Detekce objektů v pohybu ve své podstatě spočívá v segmentaci obrazu na pozadí (část obrazu, která zůstává statická) a popředí (část obrazu, která představuje pohybující se objekty). Popředí je většinou reprezentováno bitovou maskou, kterou je možné ze zpracovávaného snímku extrahovat pouze pohybující se objekty. Při psaní této kapitoly bylo čerpáno z prací [1],[2],[3] a [4].

2.1 Metody odečítání pozadí

Metody odečítání pozadí (*background subtraction*) jsou k řešení daného problému velmi často využívané. Jsou snadno implementovatelné a rychlé, tudíž je lze použít na detekci objektů v reálném čase. Principem všech těchto metod je vytvoření modelu pozadí scény a následná diference každého snímku s modelem pozadí. Za popředí jsou následně prohlášeny ty obrazové body (*pixely*) snímku, které se od modelu pozadí liší o nastavenou prahovou hodnotu, viz vzorec (2.1).

$$F(x,y) = |I(x,y) - B(x,y)| \geq T \qquad (2.1)$$

kde $I(x,y)$ představuje hodnotu obrazového bodu na souřadnicích x, y vstupního snímku, $B(x,y)$ představuje hodnotu obrazového bodu na souřadnicích x, y modelu pozadí scény, T je hodnota prahu a $F(x,y)$ je logická hodnota určující, zda $I(x,y)$ je součástí popředí snímku I.

Jednotlivé metody se rozdělují podle toho, zda v průběhu detekce používají neměnný (statický) model pozadí, nebo model pozadí přizpůsobují změnám ve scéně (dynamický model pozadí). Metody s dynamickým pozadím se dále dělí na nerekurzivní a rekurzivní v závislosti na tom, zda ke změně pozadí využívají buffer s historií posledních N snímků, nebo zda pozadí aktualizují rekurzivně po zpracování každého snímku. Obecná ukázka fungování metody odečítající pozadí je na Obrázku 2.1. [1]

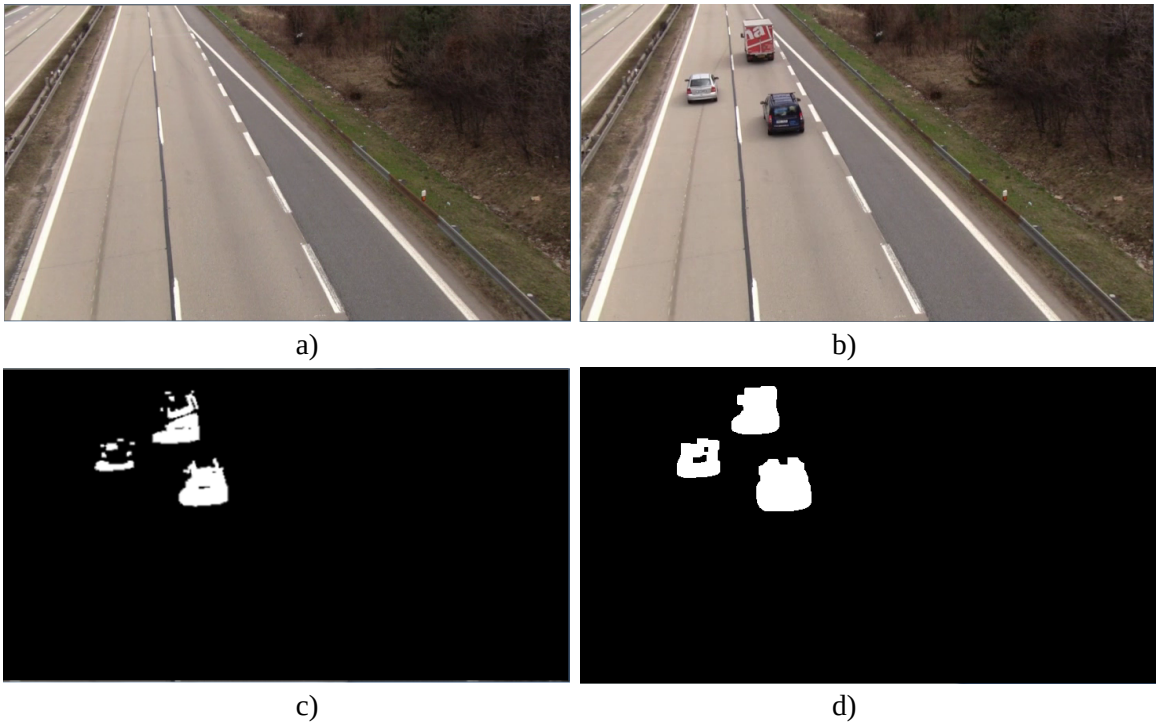
2.2 Statické modely pozadí

Nejjednodušší možná implementace metody odečítání pozadí je diference referenčního snímku, který je považován za pozadí scény, s každým snímkem videosekvence. Tato implementace není reálně příliš použitelná, protože referenční snímek musí být dopředu připravený a nesmí se na něm vyskytovat žádné objekty, které nejsou součástí pozadí. Vzhledem k tomu, že je tento snímek neměnný, tak se nedokáže přizpůsobit žádným změnám ve scéně, ať už se jedná o pouhou změnu

osvětlení, nebo drobné vychýlení kamery oproti referenční poloze, které vyústí v detekci pohybu v celém záběru.

Dalším problémem je vznik duchů, k čemuž dochází poté, co objekt, který byl součástí pozadí (například zaparkovaný automobil), opustí scénu. Na jeho místě je neustále detekován pohyb, přestože k žádnému nedošlo. Stejně tak naopak, pokud se ve scéně objeví automobil a zaparkuje (stane se tedy součástí pozadí, protože se již nepohybuje), tak bude stále detekován jako pohybující se objekt.

Vylepšením této implementace je získání referenčního snímku v učící fázi detektoru, kdy se nejdříve určitou dobu průměrují snímky videosekvence, aniž by probíhala detekce pohybu. Tímto je získaný referenční snímek pozadí scény, který dále zůstává statický. Jedinou výhodou tohoto vylepšení je tedy automatické vytvoření referenčního snímku, ze kterého jsou odfiltrovány pohybující se objekty. [2]

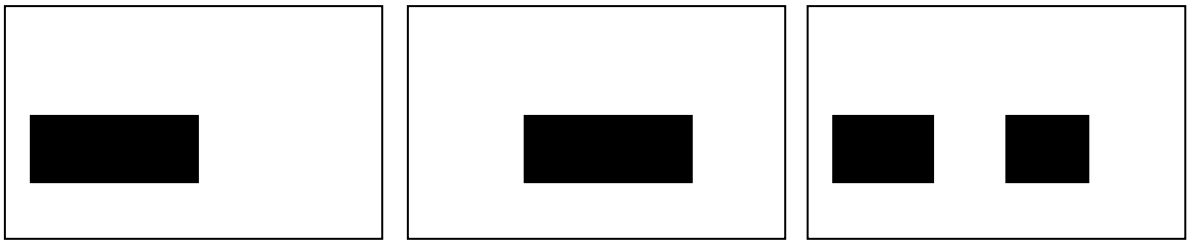


Obrázek 2.1: Ukázka metody odečítání pozadí, a) model pozadí, b) aktuální snímek, c) binární maska popředí, d) binární maska popředí po provedení optimalizací

2.3 Historie posledních *N* snímků

Metody založené na uchovávání historie posledních *N* snímků v bufferu pracují s dynamickým pozadím a jsou nerekurzivní. Nejjednodušší možná implementace je založena na diferenci dvou po sobě jdoucích snímků (historie 1 snímku), kdy je za pozadí vždy považován snímek předcházející aktuálnímu snímku. Tato implementace se vyznačuje nejvyšší možnou adaptibilitou modelu pozadí, což má ale spoustu nevýhod. Předně je tato metoda velmi náchylná na šum v obraze a špatně detekuje pomalu se pohybující objekty. Dále objekt po detekci okamžitě vstoupí do pozadí a není jej tedy možné správně identifikovat. Jako pohybující se objekt je totiž nesprávně označeno i místo

v pozadí, které představuje předchozí pozici objektu. Pokud je pohybující se objekt navíc jednobarevný, tak překrývající se části objektu v po sobě jdoucích snímcích budou mylně označeny za pozadí, což způsobí rozdělení pohybujícího se objektu na dvě části, viz Obrázek 2.2.



a) b) c)

Obrázek 2.2: Ukázka diference dvou po sobě jdoucích snímků: a) model pozadí (snímek v čase t-1), b) aktuální snímek (snímek v čase t), c) popředí snímku - objekt je rozdělen na dva pohybující se objekty, levý objekt je duch z předchozího snímku.

V bufferu je možné uchovávat více než jeden snímek a pozadí aproximovat například průměrnou hodnotou každého pixelu ze všech snímků v bufferu. Čím je tento buffer větší, tím je pomalejší adaptibilita pozadí a také se zvyšují nároky na paměť a výkon procesoru. Nicméně rozšířením bufferu lze eliminovat nežádoucí jev znázorněný na Obrázku 2.2.

Dalším možným vylepšením je pozadí neurčovat průměrováním snímků, ale pro každý pixel použít vždy medián hodnot pixelů na stejných pozicích ze snímků v bufferu. Určení mediánu je výpočetně náročnější, protože je nutné hodnoty nejdříve seřadit, ale výsledky jsou lepší díky jednoznačnosti pozadí, kdy dochází ke skokovým, nikoliv pozvolným změnám. Například pokud automobil zaparkuje a stane se statickým objektem v pozadí. Pozadí bude stále představovat vozovka, dokud se nebude zaparkované vozidlo vyskytovat alespoň v polovině snímků v bufferu, poté bude považováno za pozadí stojící vozidlo. Nedojde tedy k situaci, že pozadí je určeno průměrem mezi vozovkou a vozidlem, což neodpovídá žádnému reálnému objektu. [2]

2.4 Running Gaussian Average

Tato metoda je rekurzivní, nepracuje tedy s bufferem, ale v každém snímku postupně aktualizuje model pozadí (z čehož plyne nižší paměťová náročnost všech rekurzivních metod). Každý pixel pozadí je v tomto případě modelován Gaussovou křivkou. Ta představuje normální rozdělení pravděpodobnosti $\eta(\mu, \sigma^2)$ určující, zda hodnota pixelu aktuálního snímku patří do pozadí. Jinak bude pixel považován za součást popředí. Střední hodnota normálního rozdělení μ a rozptyl σ^2 jsou dynamicky aktualizovány podle následujících vztahů:

$$\mu_{t+1} = \alpha I_t + (1-\alpha)\mu_t \qquad (2.2)$$

$$\sigma^2{}_{t+1} = \alpha(I_t - \mu_t)^2 + (1-\alpha)\sigma_t^2 \qquad (2.3)$$

kde I_t je hodnota daného pixelu v aktuálním snímku a $\alpha \in (0{,}0;\ 1{,}0)$ je konstanta určující adaptibilitu modelu pozadí. Čím je α menší, tím pomaleji nové objekty v popředí vstupují do pozadí. Naopak větší hodnota α znamená, že se model bude rychle přizpůsobovat změnám ve scéně.

Pro rozhodnutí zda je pixel součástí popředí scény se používá vzorec 2.4:

$$F_t(x,y) = |I_t(x,y) - \mu_t(x,y)| \geq k\sigma_t \qquad (2.4)$$

kde k je konstanta určující přesnost detekce pozadí. Běžně se používá $k = 2{,}5$, protože v případě normálního rozdělení pravděpodobnosti leží 99% jím určených hodnot v intervalu $(\mu - 2{,}5\sigma;\ \mu + 2{,}5\sigma)$.

Největší výhodou metody *Running Gaussian Average* je dynamické lokální prahování, tzn. hodnota prahu určujícího, zda je pixel označen za pozadí nebo popředí scény, se mění v čase a zvlášť pro každý pixel. Citlivost detekce popředí je tedy v jednotlivých částech scény různá. Nevýhodou může být až příliš nízká citlivost detekce v oblasti scény, kde dochází k častým změnám (velký rozptyl modelu pozadí σ^2). Může být tedy vhodné stanovit nejvyšší možnou hodnotu, kterou může σ^2 nabývat. [3]

2.5 Mixture of Gaussians

Mixture of Gaussians (směsice Gaussových křivek, dále v textu označována jako *MoG*) je multimodální (pracuje současně s více modely pozadí) metoda detekce popředí popsaná v práci [4], ze které vychází následující text.

Principem metody je modelování pozadí každého pixelu pomocí K Gaussových křivek. V práci je zmiňováno, že se nejčastěji používá tři až pět křivek. Pravděpodobnost, že je pixel X zpracovávaného snímku v čase t pozadím, je vyjádřena vztahem 2.5:

$$P(X_t) = \sum_{i=1}^{K} \omega_{i,t}\eta(X_t, \mu_{i,t}, \Sigma_{i,t}) \qquad (2.5)$$

kde K je počet křivek, $\omega_{i,t}$ je váha i-té křivky v čase t, $\mu_{i,t}$ je střední hodnota i-té křivky v čase t, $\Sigma_{i,t}$ je kovariační matice i-té křivky v čase t a kde η je funkce hustoty pravděpodobnosti:

$$\eta(X_t, \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(X_t-\mu_t)^T\Sigma^{-1}(X_t-\mu_t)} \qquad (2.6)$$

Z výkonnostních důvodů autoři definují kovariační matici podle vzorce 2.7:

$$\Sigma_{k,t} = \sigma_t^2 I \qquad (2.7)$$

kde I je jednotková matice. Tato definice předpokládá, že jednotlivé barevné složky pixelu jsou na sobě nezávislé a vede při malé ztrátě přesnosti k vyšší rychlosti metody.

Hodnota pixelu X_t aktuálního snímku je porovnávána podle vzorce 2.4, kde $k = 2{,}5$, se všemi Gaussovými křivkami, které modelují pozadí daného pixelu, přičemž je zjišťována první shoda, nikoliv nejlepší shoda. Pokud není nalezena žádná shoda, je pixel zakomponován do popředí scény a zároveň je křivka s nejmenší váhou nahrazena novou křivkou se středního hodnotou odpovídající hodnotě pixelu X_t a počátečním vysokým rozptylem.

Váhy všech křivek jsou po zpracování každého snímku aktualizovány podle vzorce 2.8:

$$\omega_{k,t} = (1-\alpha)\omega_{k,t-1} + \alpha M_{k,t} \qquad (2.8)$$

kde α je učící konstanta (vyjadřuje míru adaptibility pozadí) a $M_{k,t}$ je 1 pro křivku, ve které byla nalezena shoda s pixelem X_t a 0 pro ostatní křivky. Po provedení aktualizací se musí váhy normalizovat.

Střední hodnota a rozptyl křivek, které se neshodují s pixelem X_t jsou zachovány. V případě křivky, u které byla nalezena shoda, se tyto parametry aktualizují podle vzorců 2.9 a 2.10:

$$\mu_t = (1-\rho)\mu_{t-1} + \rho X_t \qquad (2.9)$$

$$\sigma^2{}_t = (1-\rho)\sigma^2_{t-1} + \rho(X_t-\mu_t)^T(X_t-\mu_t) \qquad (2.10)$$

kde

$$\rho = \alpha\eta(X_t|\mu_k,\sigma_k) \qquad (2.11)$$

Největší výhoda tohoto řešení se projeví v situaci, kdy se nový objekt stane součástí pozadí, protože stále existují původní modely pozadí, jen je snížena jejich váha. Pokud objekt scénu následně opustí, relativně rychle se nejvyšší váha přesune na předchozí validní model pozadí.

Jednotlivé křivky jsou seřazeny podle poměru jejich váhy a rozptylu (ω/σ). To znamená, že čím je křivka v modelu pozadí déle (má vyšší váhu) a zároveň má nejnižší variabilitu (menší rozptyl), tím vyšší je její priorita. Díky tomu nebude na prvním místě křivka, která má sice vysokou váhu, ale příliš velký rozptyl a tím pádem se bude shodovat s velkým rozpětím hodnot.

Autoři dále zavádí výkonnostní vylepšení, kdy je na shodu testována pouze omezená podmnožina křivek B, která je vybrána na základě vzorce 2.12:

$$B = argmin_b(\sum_{k=1}^{b}\omega_k > T) \qquad (2.12)$$

kde T je míra minimální části dat, která mají být začleněna do pozadí. To znamená, že do podmnožiny B budou zahrnovány jen nejpravděpodobněji odpovídající křivky, dokud nebude do pozadí započítána určitá část nových hodnot pixelu. Pokud je T malé, tak je model většinou unimodální. Naopak příliš vysoká hodnota T vede k tomu, že se vždy vyberou všechny nebo téměř všechny křivky. Vhodně nastavený parametr T umožňuje modelovat pozadí části scény, kde nedochází k pohybu unimodálně a naopak části scény, kde je pohyb častý multimodálně.

Metoda *MoG* díky multimodálnímu přístupu dokáže uchovávat několik modelů pozadí pro každý pixel, což je výhodné především pokud v pozadí scény dochází k cyklickým změnám, které není žádoucí detekovat jako popředí (pohyb listů stromu ve větru, změna barvy na semaforu, měnící se reklamní panel atd.). Dále se dokáže dynamicky přizpůsobit změnám ve scéně, jakými je například střídání dne a noci, změna intenzity osvětlení scény (viz Obrázek 2.3) nebo změna objektů, které představují pozadí. *MoG* tedy lze označit za vhodnou metodu pro použití ve venkovní scéně.

Obrázek 2.3: Změna světelných podmínek ve scéně

3 Sledování pohybujících se objektů ve videosekvenci

Výstupem metod detekujících popředí je binární maska označující místo pohybu v obraze vytvořená pro každý snímek videosekvence. V této masce je nutné identifikovat jednotlivé spojité oblasti, které by mohly představovat reálné pohybující se objekty. Tyto oblasti jsou nazývány bloby a jsou reprezentovány svou pozicí (centroidem – geometrickým středem), velikostí, barvou a případně i dalšími údaji, například rychlostí a směrem pohybu. V ideálním případě je každý blob obrazem reálného objektu ve scéně. V praxi tomu tak ale většinou není, reálný objekt se může v popředí rozpadnout na více blobů (problém segmentace), nebo může být více reálných objektů spojeno do jednoho blobu (*occlusion*). Informace v této kapitole byly čerpány z prací [5], [6] a [7].

Segmentace je většinou důsledkem přítomnosti stacionárního objektu v pozadí, který se nachází mezi kamerou a pohybujícími se objekty, například lampy pouličního osvětlení. K segmentaci může také docházet i v případě, kdy má pohybující se objekt barvu velmi podobnou pozadí (šedé vozidlo splývající se silnicí). Potom detektor může zachytit pouze pohyb barevně odlišných částí vozidla, například dvou červených koncových světel. Tento problém je řešitelný spojováním blízkých blobů, kdy lze předpokládat, že se jedná o jeden reálný objekt.

Occlusion může vznikat díky úhlu záběru a perspektivě scény, kdy bližší objekty překryjí vzdálenější objekty. Může se jednat i o reálný kontakt obou objektů (například lidé držící se za ruce), což v případě aplikace pro sledování dopravy není problém, naopak třeba vozidlo s přívěsem je žádoucí detekovat jako jeden blob. Dalším původcem problému splynutí více objektů do jednoho blobu mohou být stíny objektů, které objekty vzájemně propojí. Řešení tohoto problému se věnují samostatné práce, například [8].

3.1 Přiřazování blobů k objektům a jejich sledování

Prvním problémem je mapování blobů v popředí na reálné objekty. V zásadě se jedná o provedení základní klasifikace blobů a výběr těch, které by mohly odpovídat reálným zájmovým objektům. Například pokud je cílem sledování vozidel, nebudou brány v potaz bloby, které s největší pravděpodobností vozidlo nepředstavují (jsou příliš malé nebo velké, mají odlišné proporce apod.).

Bloby, které prošly klasifikací, jsou považovány za objekty a budou sledovány v dalších snímcích. K tomu slouží různé techniky, tato práce se zaměří na techniku sledování kontur objektů. Řešeným problémem je přiřazení blobů v aktuálním snímku jim odpovídajícím objektům v předchozím snímku, přičemž objekt mezi snímky mohl kromě pozice změnit i svoji velikost a tvar. Pokud v předchozím snímku nebyl nalezen obraz blobu, bude tento blob označen za nový objekt. [5]

Samotné přiřazování blobů se nejčastěji provádí na základě matice porovnávající, nakolik si jsou bloby a objekty podobné. Například je sledováno N objektů z předchozího snímku a v aktuálním snímku je detekováno M blobů, potom bude vytvořena matice o rozměrech $N \times M$. Hodnoty v této matici značí míru podobnosti. Ta je určena pomocí Eukleidovské vzdálenosti, ve které jsou zohledněny známé vlastnosti blobů, například jejich pozice, velikost a barva, viz vzorec 3.1:

$$M_{i,j} = 1 - \sqrt{(|x_i - x_j|/f_w)^2 + (|y_i - y_j|/f_h)^2 + (|S_i - S_j|/(f_w * f_h))^2 + (|C_i - C_j|/255)^2} \quad (3.1)$$

kde x a y jsou souřadnice blobu, S je velikost blobu, C je barva blobu, f_w je šířka snímku a f_h je výška snímku. Rozdíly jednotlivých vlastností jsou normalizovány do intervalu <0; 1>. Eukleidovská vzdálenost udává míru odlišnosti blobů, proto je odečtena od 1, výsledkem je pak míra podobnosti. [7]

Každý řádek tabulky může být přiřazen maximálně jednomu sloupci, přičemž je třeba, aby součet všech přiřazených hodnot byl co nejvyšší. Nejjednodušší metodou, která je velmi rychlá, ale nemusí dávat příliš dobré výsledky, je algoritmus *Greedy search*. Ten postupně jednotlivým řádkům přiřazuje sloupec s nejvyšší mírou podobnosti. Přiřazený sloupec již není možné použít v dalších řádcích, tudíž je výsledek velmi ovlivněn pořadím řádků. V případě prvního řádku se vybírá z *N* sloupců, u druhého řádku z *N-1* sloupců a tak dále, až v posledním řádku tabulky zbývá na přiřazení pouze jeden sloupec, přestože může být naprosto nevhodný, viz Tabulka 3.1 a).

Nejlepším řešením je projít všechny permutace matice a vybrat kombinaci s nejvyšším skóre, viz Tabulka 3.1 b). Problémem je vysoká náročnost této metody na výkon, která roste spolu s počtem sledovaných a detekovaných blobů. V případě matice s rozměry 3×3, která je v ukázce, je třeba porovnat 3! $(3 * 2 * 1 = 6)$ kombinací. Pokud se ale začnou sledovat pouhé dva bloby navíc, vzroste počet možných kombinací na 120. Ani nejlepší možná kombinace ale nemusí být tou nejvhodnější, protože počítá s přiřazením každého blobu, i když bude míra podobnosti poměrně malá. Tuto metodu lze vylepšit tím, že se stanoví práh minimální míry podobnosti a pokud se na řádku nevyskytuje žádná hodnota vyšší než daný práh, nebude se tento řádek brát v potaz. Díky tomu může být zvolena vhodnější kombinace zbývajících řádků, viz Tabulka 3.1 c). [6]

	$B1_t$	$B2_t$	$B3_t$
$B1_{t-1}$	0,9	0,7	0,8
$B2_{t-1}$	0,3	0,4	0,6
$B3_{t-1}$	0,9	0,1	0,6

a)

	$B1_t$	$B2_t$	$B3_t$
$B1_{t-1}$	0,9	0,7	0,8
$B2_{t-1}$	0,3	0,4	0,6
$B3_{t-1}$	0,9	0,1	0,6

b)

	$B1_t$	$B2_t$	$B3_t$
$B1_{t-1}$	0,9	0,7	0,8
$B2_{t-1}$	0,3	0,4	0,6
$B3_{t-1}$	0,9	0,1	0,6

c)

Tabulka 3.1: Ukázka srovnávacích matic: a) algoritmus Greedy search – suma 1,6; b) nelepší možná permutace – suma 2,2; c) nejlepší možná permutace s prahem 0,7 – suma 1,7.

3.2 Predikce polohy objektu v příštím snímku

Přesnost přiřazování blobů je možné zvýšit predikováním polohy objektu v následujícím snímku na základě trajektorie jeho pohybu v předešlých snímcích. K tomuto účelu je možné využít například Kalmanův filtr, který umožňuje rekurzivně odhadovat stav neznámého lineárního dynamického

systému. Kalmanův filtr byl představen už v 60. letech a dnes existuje v mnoha různých rozšířeních. Je schopný vyčistit signál od šumu a rušení, přestože nemá dostupné žádné poznatky o šumu, který na signál působí.

Celý výpočet následujícího stavu systému v čase $t + 1$ se skládá ze dvou kroků – predikce a korekce. V prvním kroku je vytvořen odhad příštího stavu na základě posledních měření a aktuálním vstupu v čase t. Současně je odhadnuta kovariance chyby. V druhé (korekční) fázi dochází k aktualizaci odhadu na základě skutečného stavu systému v čase $t + 1$. Díky tomu se neustále zpřesňuje predikce následujících stavů. Čím déle se predikce blíží skutečnému stavu, tím má vyšší váhu, díky čemuž dochází k efektivní filtraci nežádoucího šumu.

Kromě Kalmanova filtru by bylo možné použít například i částicový filtr, což je další predikční – estimační algoritmus, který je možné použít i pro nelineární systémy. Podrobně jsou různé predikční algoritmy popsány v práci [9].

4 Návrh aplikace

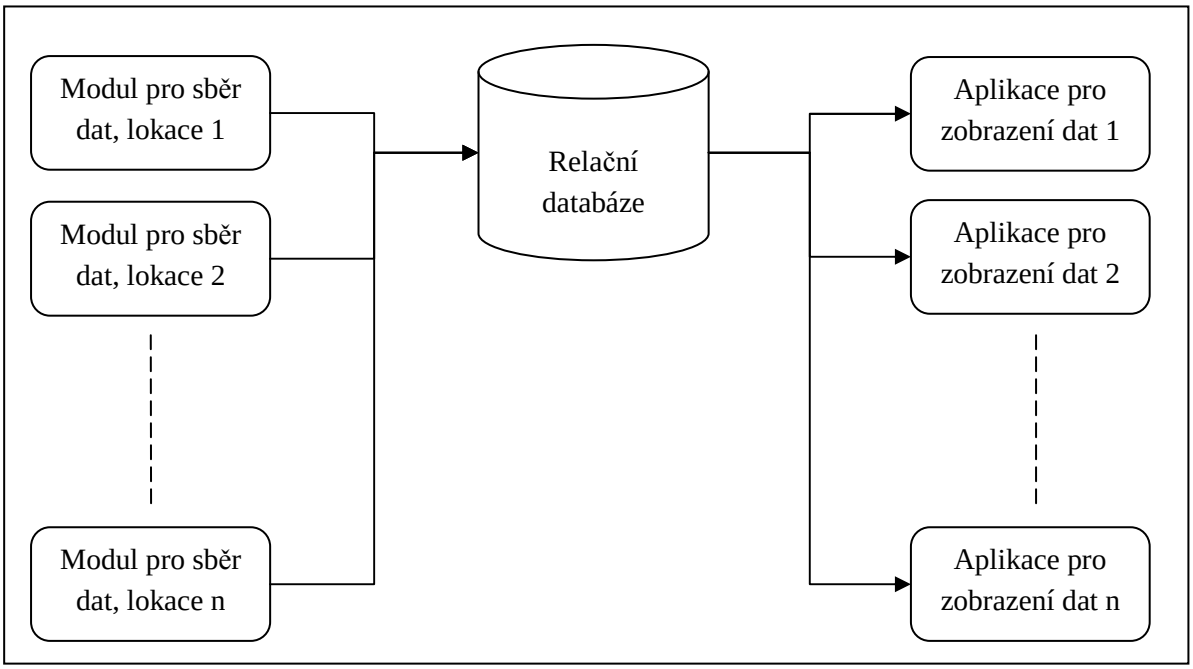
Cílem této bakalářské práce je vytvořit aplikaci, která ve videosekvenci dokáže detekovat a sledovat projíždějící vozidla a zaznamenávat o nich základní údaje, které poté budou sloužit jako vstup pro generování různých statistik o provozu. Základní vlastnosti výsledné aplikace budou následující:

- vstup – videosekvence z kamery nebo videosouboru,
- kamera – statická, umístěná nad silničním či dálničním provozem,
- výstup – grafické dopravní statistiky (grafy) pro každou kameru,
- aplikace bude:
    - zpracovávat údaje z více kamer současně,
    - zobrazovat přehled o aktuální dopravní situaci z více míst zároveň,
    - fungovat v reálném čase.

4.1 Schéma aplikace

Po analýze požadavků, které jsou na aplikaci kladeny, se jeví jako přirozené rozdělení aplikace na dvě samostatné části – detektor sbírající data o provozu (dále v textu také jako modul pro sběr dat) a aplikaci, která bude tato data vyhodnocovat, vytvářet z nich statistiky a ty zobrazovat v uživatelsky srozumitelné formě (dále v textu jako aplikace pro zobrazení statistik). V rámci práce tedy vznikne systém, ve kterém budou kooperovat dvě samostatné specializované aplikace. Toto řešení poskytuje výhodu ve formě určité modulárnosti výsledku práce, kdy bude možné detektor vyměnit za jiný při použití stejné aplikace pro zobrazení statistik a naopak. Řešení je vhodné i vzhledem k možnosti reálného nasazení systému. To předpokládá, že bude na různých místech rozmístěno více detektorů a sledování statistik bude přitom nezávislé na jejich geografickém umístění. Zároveň tím může být splněn požadavek na sjednocené zobrazení statistik ze všech instancí modulu pro sběr dat v každé instanci aplikace pro zobrazování dat.

V případě tohoto návrhu je nutné rozhodnout, jakým způsobem budou tyto dvě aplikace propojeny. Celé řešení má fungovat v reálném čase, v systému se bude vyskytovat více nezávislých detektorů a stejně tak i aplikace pro zobrazování dat nemusí být pouze jedna. Vzhledem k tomu se mi zdá jako nejlepší řešení propojit jednotlivé moduly jednou společnou relační databází. Díky relační databázi mohou všechny instance aplikace pro zobrazení statistik v reálném čase zpracovávat všechna data, která do databáze současně zapisují instance modulů pro sběr dat. Systém řízení báze dat přitom automaticky zajistí integritu databáze a správu jednotlivých transakcí tak, aby byla databáze vždy v konzistentním stavu. Grafické znázornění návrhu aplikace je na Obrázku 4.1.



Obrázek 4.1: Grafické znázornění schémata aplikace

4.2 Modul pro sběr dat

Úkolem modulu pro sběr dat je transformovat obraz z videosekvence do databázových záznamů popisujících jednotlivé průjezdy vozidel. Hlavní vlastnosti této části aplikace budou následující:

- vstup – signál z připojené kamery nebo videosoubor ve formátu avi,
- výstup – záznamy o jednotlivých vozidlech v databázi nebo XML souboru,
- modul bude:
    - detekovat a sledovat pohybující se vozidla ve scéně,
    - každému vozidlu určovat jízdní pruh, ve kterém se při detekci nachází,
    - odhadovat reálné rychlosti vozidel,
    - plně konfigurovatelný xml souborem.

Detekce a sledování vozidel

K řešení detekce a sledování pohybujících se objektů ve videu existuje mnoho metod a algoritmů, některé z nich byly popsány v kapitole 3. Moje aplikace bude k detekci používat rekurzivní metodu odečítání pozadí nazvanou *Mixture of Gaussians*, jejíž princip je v kapitole 3 taktéž popsán. Zvolená metoda je vhodná k vytvoření modelu pozadí venkovní scény. Především se dokáže dobře vypořádat s postupnými změnami osvětlení scény (například v důsledku přirozeného střídání dne a noci nebo změny počasí) a dále nedetekuje drobné změny ve scéně, jakými je například kývání větví stromů ve větru, pohyb trávy apod. Za další důležitou vlastnost této metody lze považovat nezahrnování cyklických změn ve scéně do popředí. V případě sledování dopravy se může jednat o semafory či jiné světelné ukazatele, jejichž změnu není žádoucí detekovat. Současně je tato metoda dostatečně rychlá pro použití v reálném čase.

Aby byla detekce vozidel a posléze i jejich sledování rychlejší, bude každý snímek nejdříve převeden do odstínů šedi. Algoritmy poté pracují pouze s dvourozměrnou maticí místo trojrozměrné, čímž se snižuje složitost výpočtů. Dále bude vhodné snímky předzpracovávat mírným rozmazáním a tím zabránit například detekci obrazového šumu, dešťových kapek a eliminovat falešné detekce při drobném pohybu kamery.

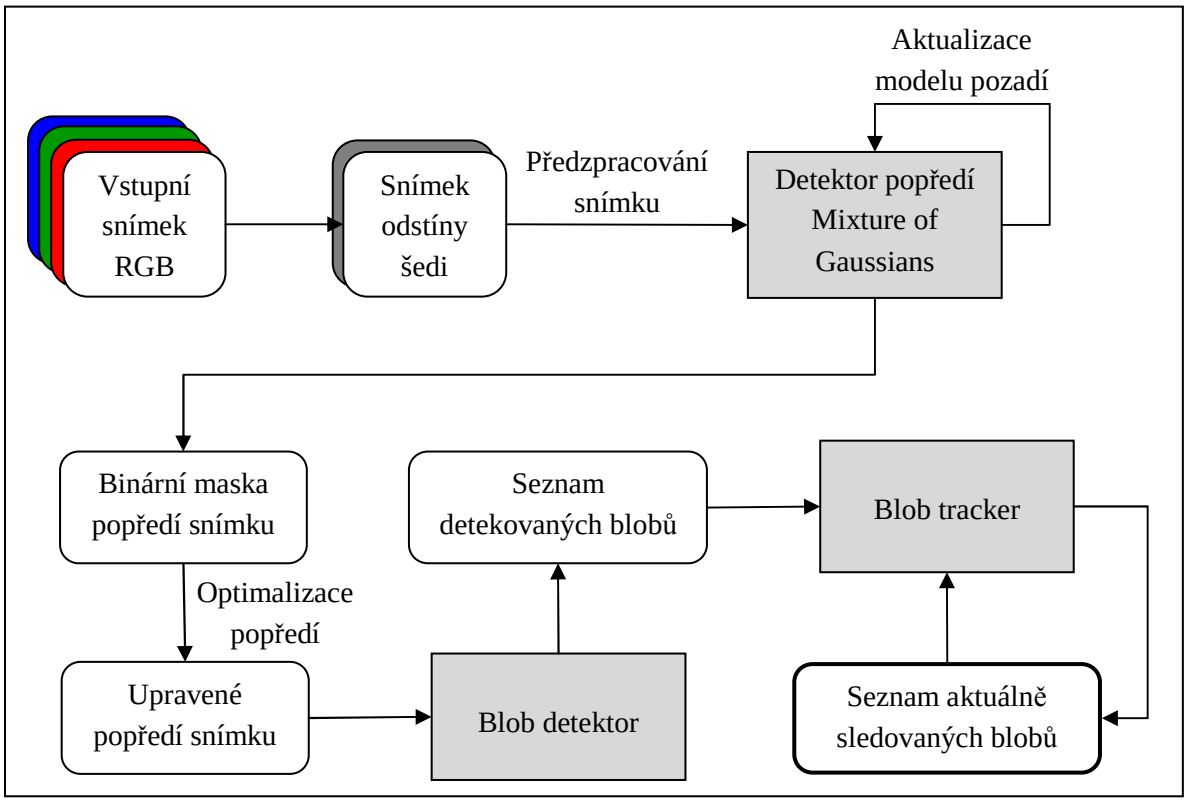
Získané popředí každého snímku bude poté nutné upravit. Pomocí masky popředí lze odstranit informace o pohybu, který leží mimo oblast zájmu. Jelikož blob v popředí náležící jednomu konkrétnímu vozidlu nemusí být vždy úplně spojitý, je vhodné provést morfologickou operaci dilatace, čímž se docílí spojení blízkých sousedících blobů a eliminuje se segmentace jednoho vozidla na více nezávislých objektů. Při této operaci může dojít k jinému nežádoucímu jevu – spojení více vozidel do jednoho blobu a tím ke ztrátě informace o některých vozidlech. Proto je po operaci dilatace dobré provést ještě její opak, erozi. Pokud se jedno vozidlo detekuje jako více blobů, tak jsou v těsné blízkosti a díky operaci dilatace se spojí do jednoho. Takto vzniklý blob již operace eroze zpětně nerozpojí. Pokud však došlo ke sloučení dvou blobů představujících dvě vozidla, tak vzhledem k větší vzdálenosti mezi těmito bloby se zpravidla pouze dotýkají malou částí své hranice, a tohle již eroze eliminovat dokáže, viz Obrázek 4.2.

Obrázek 4.2: Optimalizace popředí: původní popředí, po operaci dilatace, po operaci eroze

Výsledná binární maska popředí je vstupem detektoru blobů, který v ní nalezne všechny spojité oblasti a vrátí jejich seznam. V ideálním případě by mělo popředí snímku po provedení optimalizací obsahovat pouze ty bloby, které vždy představují právě jeden reálný objekt. Ve skutečnosti se nemusí vždy podařit popředí takto optimalizovat. Může obsahovat malé bloby, které se nepodařilo spojit dohromady, nebo naopak může v důsledku nenadálého pohybu kamery být přítomen blob o velikosti téměř celé scény. Je žádoucí, aby se tyto bloby nesledovaly v dalších snímcích, protože se pravděpodobně nejedná o reálná vozidla. Proto je třeba detekované bloby klasifikovat, v případě mé aplikace na základě minimálních a maximálních rozměrů. Konkrétně se kontroluje šířka, výška a plocha blobu. Pouze bloby, které vyhoví všem podmínkám, budou sledovány v dalších snímcích.

Sledování vozidel bude prováděno na základě porovnávání pozice a velikosti detekovaných a aktuálně sledovaných blobů. Tato metoda sledování je velmi rychlá, ale špatně se vypořádává se situací, kdy se dva samostatné objekty v popředí najednou spojí v jeden blob a ten se následně opět rozdělí. Tato vlastnost by v případě sledování dopravy kamerou umístěnou nad komunikací neměla mít na výsledek vážný negativní vliv. Ke spojování vozidel jedoucích vedle sebe příliš nedochází a spojování blobů vozidel jedoucích za sebou je těžko řešitelné, protože budou vždy splývat díky perspektivě scény.

V předchozích odstavcích byl podrobně popsán způsob detekce a sledování vozidel ve videosekvenci, celý proces je též přehledně zobrazen na Obrázku 4.3.



Obrázek 4.3: Proces zpracování vstupního snímku

Odhadování rychlosti projíždějících vozidel

Rychlost projíždějících vozidel je údaj, který je možné aproximovat i detektorem zpracovávajícím pouze videosignál bez žádného dalšího měřícího zařízení. Princip je poměrně jednoduchý. Videosignál má vždy definovaný počet snímků za sekundu (FPS, *frames per second*), tím pádem je možné určit, za jak dlouho vozidlo projelo určitý úsek ve scéně v reálném čase. Dále je k dispozici údaj o vzdálenosti, kterou vozidlo v obraze urazilo v pixelech. Zbývá tedy přepočítat pixelovou vzdálenost na reálnou vzdálenost. Toto je ovšem kvůli perspektivě ve scéně velmi náročné, ne-li nemožné.

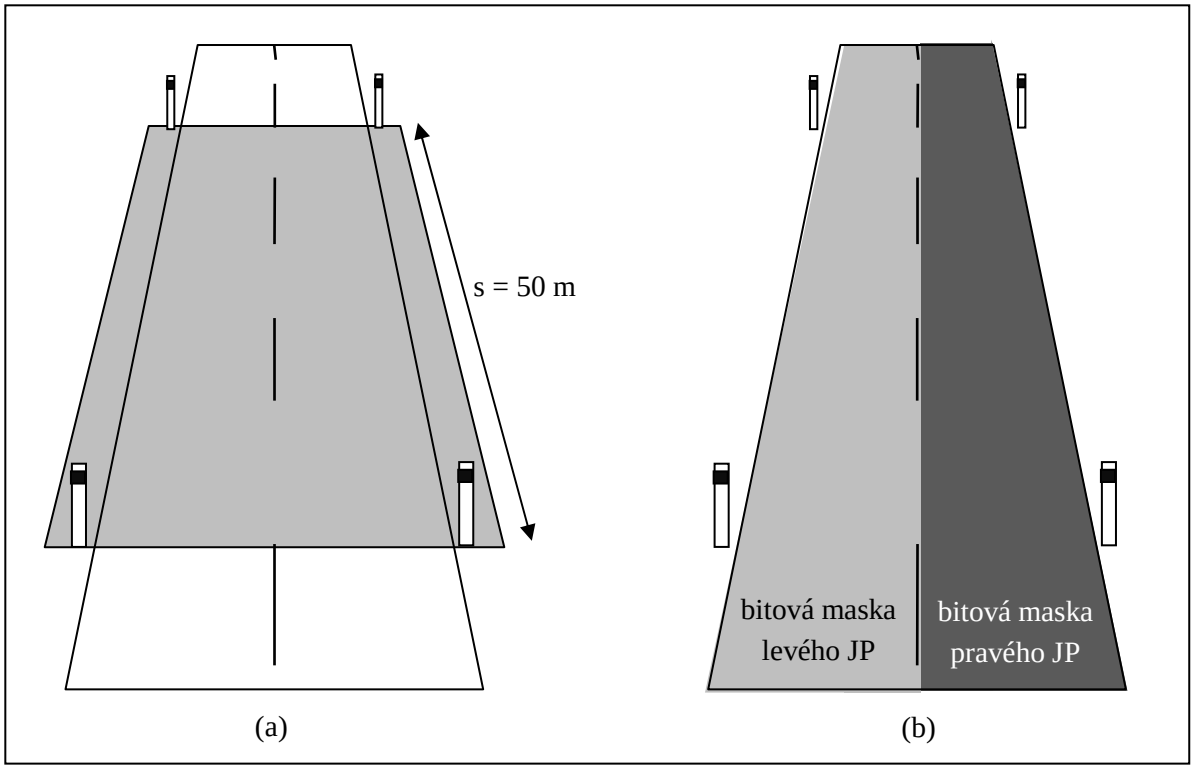
Řešení spočívá ve vyznačení určitého úseku v obraze, kterým vozidlo projíždí, přičemž je známá jeho reálná délka v metrech. Je tedy nutné, aby kamera ve scéně zachytila i reálné referenční body, mezi kterými je známá skutečná vzdálenost a které lze v obraze jednoznačně označit. Kromě speciálních referenčních bodů, které by za tímto účelem bylo možné v okolí komunikace vytvořit, lze využít i znalosti technických norem již existujícího dopravního značení. Například svislé směrové sloupky jsou podle normy *ČSN 736101* od sebe vzdáleny 50 metrů na přímé komunikaci nebo ve směrovém oblouku o poloměru větším než 1 250 metrů [10]. Ukázka takového úseku je na Obrázku 4.4. a). Z vodorovného dopravního značení je možné použít třeba podélnou čáru přerušovanou, která má v provedení *č. V 2a*, určeném pro oddělení jízdních pruhů, délku 3 nebo 6 metrů a vzdálenost mezi jednotlivými čárami je vždy rovna dvojnásobku jejich délky [11].

Při sledování vozidla je nutné kontrolovat, jestli referenční bod na vozidle vstoupil do vyznačeného úseku a pokud ano, tak si zapamatovat číslo snímku, ve kterém k tomu došlo. Poté je

potřeba kontrolovat, jestli referenční bod na vozidle již tento úsek opustil, v tom případě se vypočítá odhad rychlosti vozidla podle vzorce 4.1:

$$v = s / \frac{c_2 - c_1}{f} \qquad (4.1)$$

kde c_2 je číslo snímku, ve kterém vozidlo opustilo měřený úsek, c_1 je číslo snímku, ve kterém vozidlo vstoupilo do měřeného úseku, f je počet snímků za sekundu a s je reálná délka úseku v metrech. Výsledkem v je rychlost vozidla v metrech za sekundu.



Obrázek 4.4: a) úsek vozovky, na kterém probíhá měření rychlosti, b) rozdělení vozovky na jednotlivé jízdní pruhy bitovými maskami

Přesnost tohoto odhadu závisí na FPS a rozlišení vstupního videosignálu. Pro zvýšení přesnosti zvolené metody i při menším FPS jsem zavedl optimalizaci, která pracuje s "virtuálními půlsnímky" a teoreticky je tedy schopna podávat stejné výsledky, jako kdyby byla snímkovací frekvence dvojnásobná. Princip této optimalizace spočívá v tom, že si aplikace uchovává vždy kromě aktuální polohy referenčního bodu sledovaného vozidla i jeho polohu v předchozím snímku. Pokud referenční bod v aktuální poloze přejel hranici měřeného úseku, spočítá se jeho předpokládaná poloha ve "virtuálním půlsnímku" mezi aktuálním a předchozím snímkem, což představuje polovinu jejich vzájemné vzdálenosti. Poté se porovnává, která z těchto tří poloh má nejblíže k hranici měřeného úseku a jejíž číslo snímku se tím pádem použije při výpočtu rychlosti vozidla (může to být číslo aktuálního snímku, číslo předchozího snímku nebo dopočítaný půlsnímek). Tato optimalizace může teoreticky zdvojnásobit přesnost aproximace rychlosti vozidla, ale pouze za předpokladu, že se vozidlo pohybuje konstantní rychlostí. Nemůže se ale stát, že by se přesnost odhadu zhoršila.

Určení jízdního pruhu

Dalším údajem, který lze z obrazu získat, je jízdní pruh, ve kterém se detekované vozidlo pohybuje. Každý jízdní pruh může být vyjádřen například bitovou maskou a při detekci vozidla poté stačí zjistit, jestli se referenční bod vozidla nachází uvnitř oblasti definované maskou některého z jízdních pruhů, viz Obrázek 4.4 b). Tímto způsobem je možné sledovat i pohyb vozidla mezi pruhy, kdy se může porovnávat aktuální jízdní pruh s jízdním pruhem detekovaným v předchozím snímku a tímto zaznamenávat všechny změny jízdních pruhů během sledování vozidla.

4.3 Aplikace pro zobrazení statistik

Součástí navrhnutého systému je aplikace, která bude schopná z dat, jež jsou výstupem modulu pro sběr dat popsaného v předchozí podkapitole, vytvořit souhrnné statistiky a zobrazit je uživateli. Tato část aplikace bude mít následující vlastnosti:

- vstupem budou data z databáze,
- aplikace z dat vytvoří souhrnné agregované statistiky za zvolené časové období,
- statistiky ve formě grafů zobrazí uživateli,
- aplikaci bude mít přehledné uživatelské rozhraní

Rozhodl jsem se tuto část systému navrhnout jako webovou aplikaci, protože pro účely vytváření a zobrazování statistik se mi jeví jako vhodnější. V databázi jsou uchovávány záznamy o každém vozidle, které se podařilo úspěšně detekovat, což představuje poměrně velké množství dat. V případě klasické desktop aplikace by zobrazení statistik představovalo přenos všech těchto dat z databáze do klientské aplikace, následné vytvoření souhrnných statistik a poté jejich zobrazení. Tento proces by se musel opakovat vždy pro každou instanci aplikace, což by reálně znamenalo pro každého uživatele. Důsledkem toho by docházelo k častému vytížení databáze. Další nevýhodu by představovala nutnost instalace aplikace, což je v dnešní době rozmachu internetových služeb pro uživatele zbytečně obtěžující.

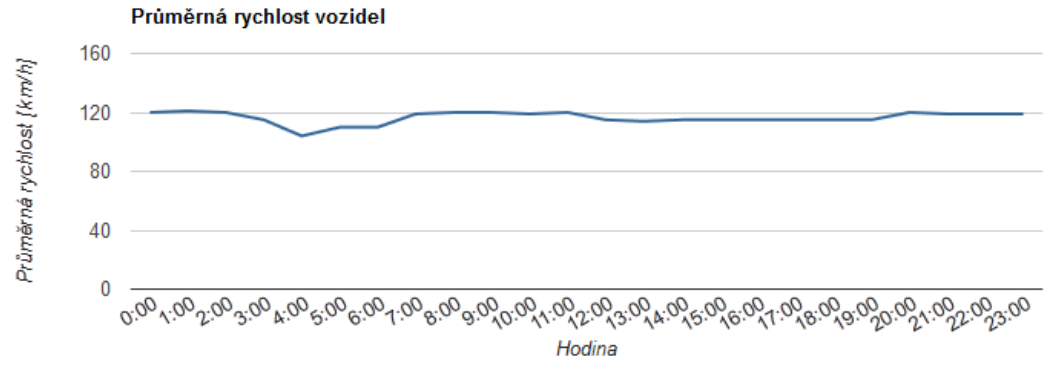
Do webové aplikace se samozřejmě musí přenést stejné množství vstupních dat, nicméně již vytvořené souhrnné statistiky se mohou uchovávat na serveru a jedna instance této aplikace dokáže současně obsloužit velké množství uživatelů, kteří mají okamžitý přístup k vytvořeným statistikám. Pokud bude uživatel chtít zobrazit statistiky, které na serveru nejsou k dispozici, tak si pro ně webová aplikace vyžádá potřebná data z databáze, statistiky vytvoří a výsledek následně uloží. Další přístup třeba i jiného uživatele k této statistice bude již okamžitý a bez dalších nároků na přenášení dat z databáze nebo na procesorový čas serveru nutný pro jejich vytvoření.

Typy podrobných statistik

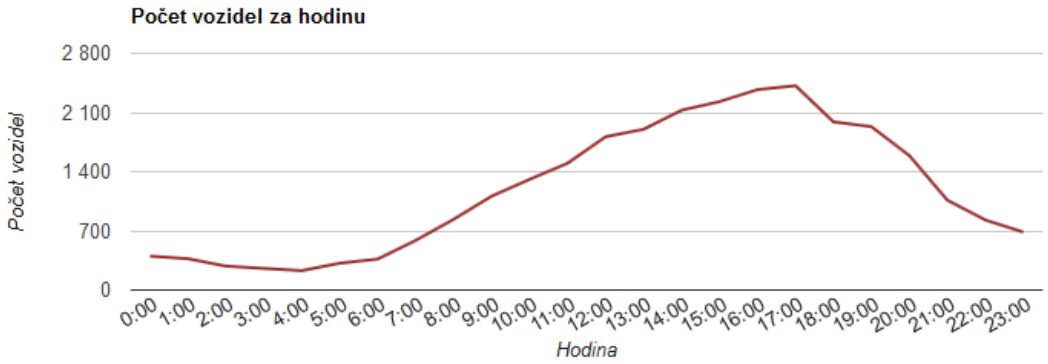
O každém detekovaném vozidle bude dostupné datum a čas detekce, odhad jeho rychlosti a informace o jízdním pruhu, ve kterém se vozidlo nacházelo na začátku a na konci sledování. Těchto několik údajů stačí na vytvoření různých souhrnných statistik pro každou lokaci, může to být například:

- průměrná rychlost a počet vozidel v určitých časových intervalech v rámci celé lokace, případně tyto statistiky mohou být rozdělené pro jednotlivé jízdní pruhy,
- kolik vozidel změnilo na sledovaném úseku jízdní pruh,
- v případě umístění kamery na dálničním nájezdu, by bylo možné určit, kolik vozidel se na daném místě na dálnici napojilo a jakou se pohybovala průměrnou nebo nejčastější rychlostí,
- kolik vozidel bylo detekováno v odstavném jízdním pruhu,
- kolik vozidel porušilo na sledovaném úseku nejvyšší nebo nejnižší povolenou rychlost,
- v případě umístění kamery v křižovatce, by bylo možné sbírat data o vytíženosti jednotlivých cest a podle toho upravit intervaly světelných signálů semaforů.

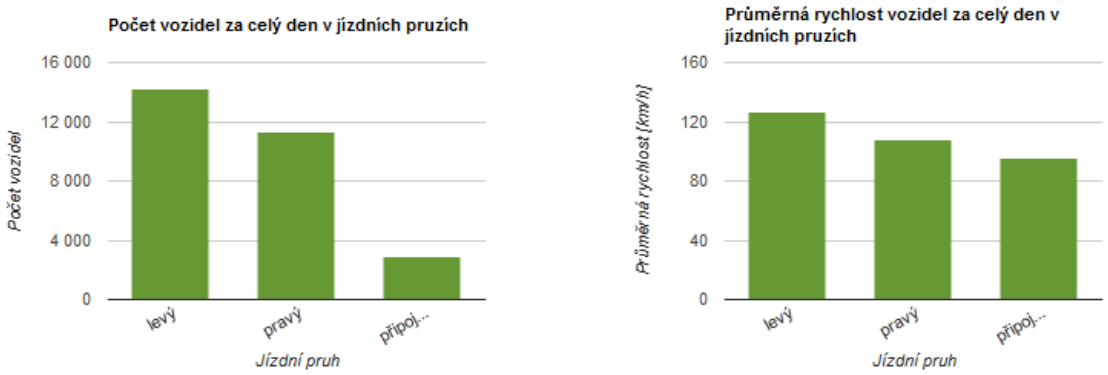
Výsledná implementace aplikace bude umožňovat vytvoření souhrnných hodinových statistik průměrných rychlostí (viz Obrázek 4.5) a počtu vozidel v dané lokaci (Obrázek 4.6) a denní statistiky průměrných rychlostí a počtu vozidel v jednotlivých jízdních pruzích sledované lokace, viz Obrázek 4.7.



Obrázek 4.5: Hodinové statistiky průměrných rychlostí vozidel



Obrázek 4.6: Hodinové statistiky počtu projetých vozidel



Obrázek 4.7: Denní statistiky pro jednotlivé jízdní pruhy (počet vozidel a průměrná rychlost)

Přehled o aktuálním stupni provozu

Kromě možnosti zobrazení podrobných statistik každého sledovaného úseku, bude výsledná aplikace také umožňovat uživatele přehledně informovat o aktuálním stupni provozu na všech lokacích současně. Aktuální stupeň provozu bude v tomto případě reflektovat průměrnou rychlost vozidel, které danou lokací projely za posledních 15 minut. Průměrná rychlost je k určení stupně provozu v tomto případě vhodnější než počet projetých vozidel za daný časový okamžik. Pokud nastane na komunikaci nějaký problém a dojde ke tvoření kolon, bude počet vozidel klesat, což by běžně značilo zlepšování dopravní situace (menší provoz), přestože tomu bude naopak. Snižující se průměrná rychlost reflektuje vznikající problém správně. Jestliže je průměrná rychlost dostatečně vysoká, lze dopravu na daném úseku označit za plynulou, nehledě na počet vozidel, který může být i v tomto případě vysoký.

Další možnosti integrace s modulem pro sběr dat

Tato podkapitola se věnuje dalším potencionálním možnostem, které navrhnutá architektura výsledné aplikace nabízí, ale které v rámci této bakalářské práce nebudou implementovány.

V relační databázi mohou být uložena také veškerá nastavení jednotlivých modulů pro sběr dat. Tato nastavení by mohla být ovlivnitelná z prostředí webové aplikace, která by byla nástrojem pro uživatelsky přívětivou vzdálenou správu modulů. Každý modul by cyklicky zjišťoval, zda není v databázi nové nastavení, které by poté ihned použil namísto aktuálního. Pro minimalizaci přenášených dat by stačilo, kdyby se modul dotázal na čas poslední aktualizace nastavení a samotná data by se přenášela pouze v případě změny.

Webová aplikace by také mohla zobrazovat obraz z kamery připojené k modulu, protože do relační databáze lze uložit i binární data. Nejednalo by se o video stream, ale o reálný pohled na dopravní situaci aktualizovaný například jednou za minutu.

5 Implementace aplikace

Výsledná aplikace je rozdělena na dva samostatné celky, podrobnosti jejich implementace budou popsány v samostatných podkapitolách, modul pro sběr dat v podkapitole 5.2 a webová aplikace v podkapitole 5.3. Obě části jsem implementoval v prostředí *Microsoft Visual Studio 2010*.

5.1 Použité technologie a knihovny

V této podkapitole budou krátce uvedeny technologie a knihovny, které jsem při řešení použil.

Knihovna OpenCV

OpenCV (*Open Source Computer Vision Library*) je knihovna určená pro práci s obrazem. Využití nachází především v aplikacích zabývajících se počítačovým viděním a zpracováním obrazu v reálném čase. Knihovna je poskytována zdarma pod licencí BSD včetně zdrojových kódů, i pro komerční využití. Je multiplatformní (kromě systémů Windows a Linux podporuje i MacOS a Android). Je napsána v jazyce C, ale je možné ji použít i v jiných programovacích jazycích (od verze 2.0 má knihovna také *C++* rozhraní a dále existují její wrappery pro jazyky *Python*, *C#* a *Java*). [12]

Aktuální verze je 2.3.1, která je také využita v implementaci této bakalářské práce (ke stažení pro Windows zde: http://sourceforge.net/projects/opencvlibrary/files/opencv-win/2.3.1).

Microsoft .NET Framework

Microsoft .NET Framework je platforma pro vývoj a běh aplikací a webových služeb. Její architektura je zobrazena na Obrázku 5.1. Informace byly čerpány z [13].

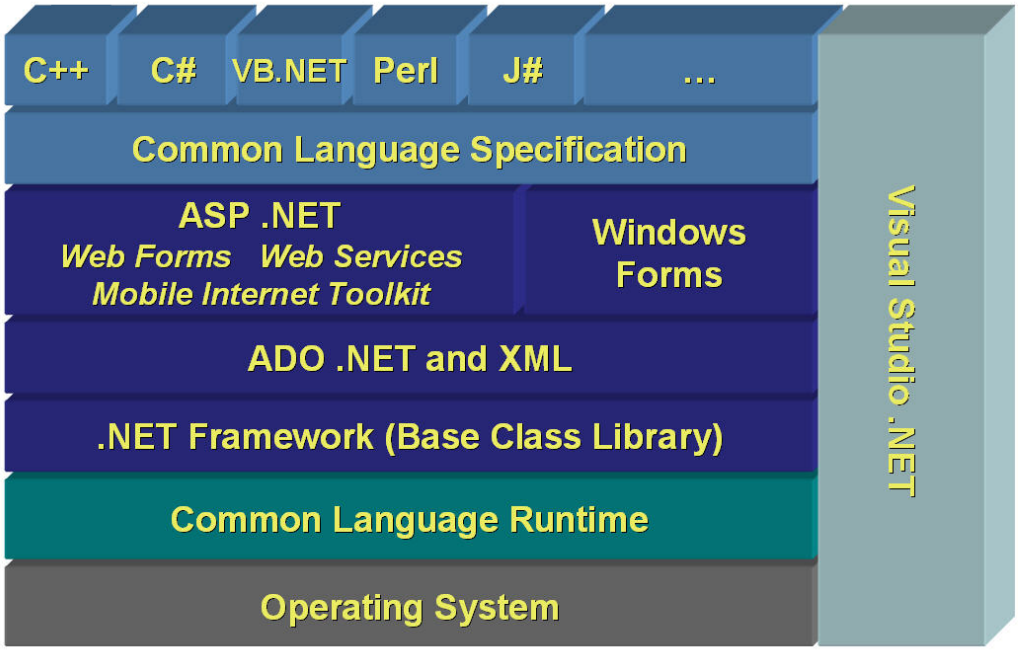
Nejnižší vrstvou je *Common Language Runtime* (*CLR*), což je spouštěcí engine aplikací. Provádí *just-in-time* kompilaci mezikódu (známého jako *Common Intermediate Language*) do strojového kódu a má na starost správu zdrojů, přidělování paměti, řízení životního cyklu objektů a zajišťuje bezpečnost běžících aplikací (aplikace nemohou přímo přistupovat k paměti). Přínosem *CLR* je přenositelnost mezikódu, kdy se programátor aplikace nemusí ohlížet na vlastnosti cílové architektury procesoru a operačního systému. *CLR* lze nejlépe přirovnat k *Java Virtual Machine*.

Nad *CLR* jsou základní knihovny .NET Frameworku, které poskytují širokou funkcionalitu od práce se soubory, vstupními a výstupními proudy, XML dokumenty, až po podporu vytváření aplikací lokalizovaných do více jazyků a jednoduchý přístup k datovým záznamům uloženým v různých databázích (*ADO.NET*).

Pomocí .NET Frameworku lze vytvářet klasické desktop Windows aplikace (s použitím *Windows Forms* nebo *Windows Presentation Foundation*) nebo webové aplikace díky *ASP.NET*, které se snaží přenést principy uplatňované při vývoji desktop aplikací (událostmi řízené programování) do vývoje internetových aplikací. To je umožněno díky takzvanému *ViewState*, který

vytváří stavové prostředí nad bezstavovým HTTP protokolem, přes který probíhá komunikace mezi klientem a serverem.

Samotný vývoj aplikací může probíhat v různých programovacích jazycích, které jsou s .NET kompatibilní, to znamená že kód v nich napsaný je přeložitelný do mezikódu, který interpretuje *CLR*. Nejpoužívanějšími jazyky jsou *C#* a *Visual Basic .NET*.



Obrázek 5.1: Architektura .NET Framework, převzato z http://srinivasbn.blogspot.com

LINQ

LINQ (*Language Integrated Query*) je součástí .NET Frameworku od verze 3.5. Jedná se o jazyk podobný *SQL* používaný k dotazování nad různými daty, které mohou být uloženy v databázi, XML souboru nebo paměti. Je tedy možné pracovat například s polem prvků velmi podobně, jako kdyby se jednalo o záznamy v databázi. Data lze například filtrovat a třídit na základě tzv. lambda výrazu, což je zkrácený zápis anonymní funkce, jejímž výsledkem je pravdivostní hodnota, na základě které bude prvek do výsledku zařazen. [14]

MySQL a knihovny pro komunikaci s databází

MySQL je multiplatformní relační databázový systém ovládaný příkazy jazyka SQL (*Structured Query Language*). Původně jej vyvinula švédská společnost *MySQL AB*, která je momentálně vlastněna společnosti *Oracle Corporation*. K dispozici je pod bezplatnou GPL i komerční licencí. Tento databázový systém je velmi oblíben především v open source komunitě (používají jej například nejrozšířenější redakční systémy *Joomla*, *WordPress* a *Drupal*). Aktuální verze je 5.5, přičemž moje aplikace pracuje s verzí 5.0 a vyšší (až ve verzi 5.0 byla přidána podpora databázových triggerů). [15]

V implementaci výsledné aplikace jsou použity volně šiřitelné knihovny určené pro práci s *MySQL* databází. Modul pro sběr dat využívá oficiální *Connector/C libmysql* (ke stažení zde: http://www.mysql.com/downloads/connector/c) a nad ním C++ nadstavbu *MySQL++* vyvíjenou

třetí stranou (ke stažení zde: http://tangentsoft.net/mysql++). Webová aplikace používá oficiální *Connector/NET* (ke stažení zde: http://www.mysql.com/downloads/connector/net).

Javascriptové knihovny

jQuery je open source knihovna volně šiřitelná pod licencí GNU GPL nebo MIT, původně vyvinuta Johnem Resigem v roce 2006. Dnes se jedná o nejpoužívanější javascriptovou knihovnu. Její hlavní přínos je v oddělení funkčnosti od struktury stránky, stejně jako CSS styly oddělují vzhled. Umožňuje velmi snadnou manipulaci s DOM elementy stránky, ať už se jedná o jejich změnu, přidávání nebo odebírání. Veškeré operace provedené pomocí této knihovny jsou kompatibilní mezi různými prohlížeči. Knihovna je modulární a jednoduše rozšiřitelná o novou funkcionalitu díky mnoha existujícím pluginům. [16]

Google Chart Tools je služba provozovaná společností Google, která zdarma poskytuje nástroj pro dynamické vytváření efektních grafických znázornění dat, například tabulek a různých druhů grafů za použití HTML 5.

Google Maps API je rozhraní, přes které lze integrovat a individuálně upravit Google Mapy přímo do vlastní webové stránky. Jeho použití je zdarma, pokud je mapa zobrazena maximálně 25 000 krát denně.

5.2 Modul pro sběr dat

Modul pro sběr dat jsem implementoval v jazyce C++ s využitím externích knihoven *OpenCV*, *MySQL++* a *TinyXML* jako konzolovou Windows aplikaci s možností otevření oken pro zobrazení aktuálního průběhu zpracování videosekvence. Základní vlastnosti a návrh řešení této aplikace byly popsány v kapitole 4.2.

Implementace programu

Samotná implementace programu je rozdělena do několika tříd:

- **`RunParams`** – zpracování parametrů příkazové řádky,
- **`BackgroundModel`** – zapouzdření modelu pozadí, provádí předzpracování snímku, aktualizaci pozadí, získání popředí a optimalizaci masky popředí,
- **`BlobTracker`** – zapouzdření funkcionality detekce a sledování vozidel,
- **`MeasureSystem`** – odhad rychlostí vozidel a detekce jízdních pruhů,
- **`ModuleTime`** – udržování aktuálního modulového času, který může při zpracování videosouboru plynout rychleji nebo pomaleji než reálný čas v závislosti na výkonu počítače,
- **`ModuleConfiguration`** – načítání a uchování konfigurace modulu,
- **`DataLayer`** – zapouzdření zápisu dat do databáze a XML souboru.

V hlavní smyčce programu se postupně zpracovávají jednotlivé snímky vstupní videosekvence. Každý snímek je převeden do odstínu šedi a následně je předán jako parametr veřejné metodě `BackgroundModel::ProcessForeground`, která vrátí výsledné optimalizované popředí snímku. Poté je zavolána metoda `BlobTracker::ProcessTracker`, která má jako parametry popředí snímku, původní snímek v odstínech šedi a číslo zpracovávaného snímku. Tato metoda zařídí detekci a sledování vozidel, výpočet jejich rychlostí a zápis statistik do databáze nebo výstupního XML souboru. V hlavním cyklu probíhá také detekce stisknutých kláves a vykreslování informací do otevřených oken programu. Po zpracování všech snímků videosekvence, nebo stisknutí klávesy `ESC`, dojde k uvolnění všech alokovaných zdrojů a ukončení programu. Nejdůležitější třídy programu budou popsány dále.

Třída BackgroundModel

Třída `BackgroundModel` zapouzdřuje funkcionalitu modelu pozadí implementovaného v rámci knihovny *OpenCV*. Konkrétně se jedná o vylepšený adaptivní model pozadí *Mixture of Gaussians*, jehož implementace vychází z článku [17].
Tato třída má pouze tři veřejné metody:

- **`SetFGMask`** ze seznamu bodů vytvoří masku popředí, která bude používána k vyčištění popředí od všech detekovaných objektů mimo oblast zájmu.
- **`DrawFGMask`** vykreslí aktuálně nastavenou oblast zájmu do předaného snímku.
- **`ProcessForeground`** slouží k získání popředí aktuálního snímku videosekvence předaného parametrem. Nejdříve je provedeno Gaussovo rozostření snímku (míru rozostření lze ovlivnit v konfiguračním souboru) a následně je zavolána metoda `Process` interní instance modelu pozadí *MoG*, která aktualizuje pozadí modelu. Poté je získáno popředí snímku zavoláním metody `GetMask`. Takto získané popředí se vyčistí od objektů mimo oblast zájmu a optimalizuje se provedením morfologických operací dilatace a eroze. Počet iterací obou operací je možné ovlivnit v konfiguračním souboru. Výsledné popředí je metodou `ProcessForeground` vráceno.

Třída BlobTracker

Třída `BlobTracker` provádí detekci a sledování vozidel a z videosekvence získává statistická data. K tomu využívá blob detektor a tracker implementovaný v knihovně *OpenCV* a vlastní třídu `MeasureSystem`, která bude popsána později v této kapitole. Detektor a tracker jsou v knihovně implementovány podle článku [18].

V knihovně *OpenCV* představuje každý blob struktura `CvBlob`, která obsahuje ID blobu, jeho pozici v obraze (x a y souřadnice) a velikost (šířku a výšku). To ale není vzhledem k potřebným výstupům aplikace dostatečné. Proto moje třída `BlobTracker` pracuje s rozšířenou strukturou `BlobTrackerBlob`, která obsahuje samotnou strukturu `CvBlob` a navíc další údaje, které jsou nutné pro získání podrobnějších informací o vozidle:

```
typedef struct BlobTrackerBlob
{
        CvBlob  blob;
        CvBlob  old_blob;
        double y_old;
        double line1_pass_frame;
        double line2_pass_frame;
        bool line1_pass;
        bool speed_finish;
        char *time;
        int frame;
        int lane;
        int laneEnd;
        int speed;
} BlobTrackerBlob;
```

Veškerou funkcionalitu třídy zajišťuje veřejná metoda `ProcessTracker`, které je v hlavní smyčce programu předáván každý snímek spolu se svou maskou popředí. Uvnitř této metody jsou volány postupně následující privátní metody třídy:

- **`TrackBlobs`** – Každému blobu ze seznamu aktuálně sledovaných blobů se pokusí najít jeho novou pozici v aktuálním snímku pomocí metody `ProcessBlob` implementované v *OpenCV* trackeru. Když je známá nová poloha blobu, je aktualizovaný blob předáný metodě `MeasureSystem::ProcessBlobSpeed`, která zajišťuje odhad rychlosti vozidla a bude vysvětlena později.

  Tímto je dokončeno sledování dříve detekovaných blobů a každý blob v seznamu aktuálně sledovaných blobů má přiřazenou novou pozici a velikost v aktuálním snímku.

- **`DeleteOldBlobs`** – Ze seznamu aktuálně sledovaných blobů odstraní ty bloby, které splňují kritéria pro odstranění. Bloby se přestávají sledovat, pokud je jejich šířka, výška nebo plocha menší, než je nastaveno v konfiguračním souboru. Tento postup předpokládá, že se bloby vždy postupně zmenšují, až jsou natolik malé, že se z trackeru vyloučí a nebudou již sledovány.

  Pokud je blob dostatečně velký, aby se dále sledoval, ale současně se jeho hranice již dotýkají hrany obrazu, je provedena optimalizace jeho velikosti a to tím způsobem, že je mu zmenšena šířka a výška v určitém poměru k původní velikosti. Tím je zabráněno tomu, aby blob v popředí postupně vyjel ze záběru a přitom zůstal v seznamu sledovaných blobů, protože jakmile se blob dotýká okraje obrazu, je neustále zmenšován, až nakonec zanikne.

  Kritéria pro odstranění blobu a poměr nové velikosti vůči původní při provádění optimalizace je možné nastavit v konfiguračním souboru. Při odstraňování blobu z trackeru zároveň dochází k zápisu dat do databáze nebo XML souboru zavoláním metody `DataLayer::AddRecord`.

- **`DetectNewBlobs`** – Provádí detekci nových blobů. Uvnitř této metody je zavolána metoda `DetectNewBlob` implementovaná *OpenCV* trackerem, které je předán seznam aktuálně sledovaných blobů. Vrácen je seznam všech nových blobů v záběru. Každý nově detekovaný blob musí vyhovět kritériím na svoji nejmenší a největší výšku, šířku a plochu, jinak nebude zařazen do dalšího sledování.

Hodnoty těchto kritérií se nastavují v konfiguračním souboru programu a je nutné, aby nejmenší velikost blobu zařazeného do sledování byla větší než velikost blobu, který bude z trackeru odstraněn. Jinak by mohlo dojít k odstranění a následné opětovné detekci blobu. Jestliže blob vyhověl všem požadavkům, je jeho struktura `CvBlob` obalena strukturou `BlobTrackerBlob` a ta je přidána do seznamu aktuálně sledovaných blobů. Vzniklá struktura je poté předána metodě `MeasureSystem::ProcessBlobLane`, která zjistí jízdní pruh, ve kterém byl blob detekován.

Třída MeasureSystem

Třída `MeaseruSystem` pracuje se strukturou `BlobTrackerBlob` a zajišťuje odhad rychlosti vozidla a detekci jízdního pruhu, ve kterém se vozidlo nachází. Nejdůležitější veřejné metody třídy jsou:

- **`SetMeasureSystemParams`** – Po vytvoření nové instance třídy je nutné ji inicializovat zavoláním této metody, které se předají bitové masky jednotlivých jízdních pruhů, bitová maska úseku pro měření rychlosti a údaje nutné pro výpočet rychlosti jako FPS videosignálu a reálná délka měřeného úseku.

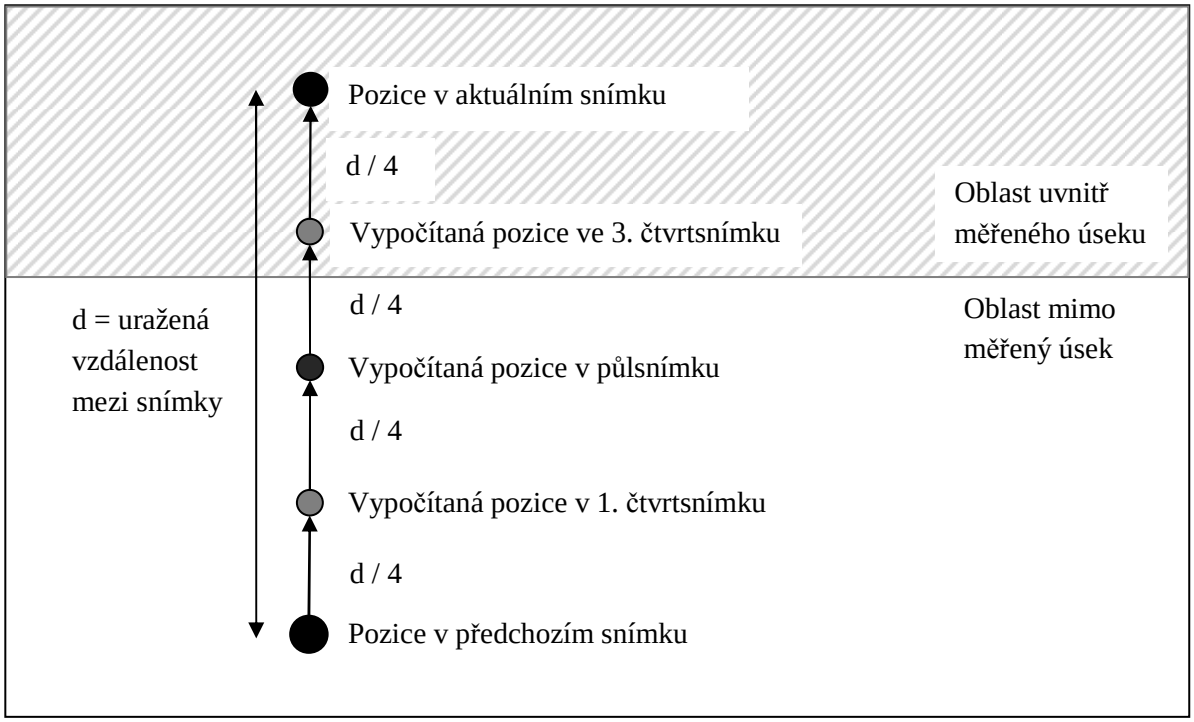
  Bitové masky jednotlivých pruhů jsou interně uloženy jako jednokanálový obraz s hloubkou 8 bitů. Tím je zajištěno, že každý obrazový bod vždy náleží nejvýše jednomu jízdnímu pruhu, který je reprezentován svým identifikačním číslem. Tímto způsobem je možné do jednoho obrazu uložit bitové masky až 255 jízdních pruhů současně. Tato optimalizace vede k vyššímu výkonu programu. Při zjišťování jízdního pruhu, který přísluší obrazovému bodu, se přistupuje do paměti pouze jednou, namísto toho, aby se sekvenčně procházely bitové masky všech jízdních pruhů.

- **`ProcessBlobLane`** – Provádí detekci jízdních pruhů. Načte hodnotu z masky jízdních pruhů na pozici, která představuje střed předaného blobu a vrátí identifikační číslo pruhu nebo hodnotu -1, pokud danému bodu nenáleží žádný jízdní pruh.

- **`ProcessBlobSpeed`** – Odhaduje rychlost pohybu předaného blobu. Struktura `BlobTrackerBlob` nese informaci o tom, jestli bylo již měření ukončeno, zda se blob nachází uvnitř měřeného úseku (a pokud ano, tak i číslo snímku, ve kterém ke vstupu došlo) a pozici blobu v minulém snímku. V kapitole 4.2 byl vysvětlen princip odhadování rychlosti, který pracuje pouze s čísly snímků, ve kterých došlo ke změně pozice blobu vůči měřenému úseku (blob vjel do úseku nebo z něj vyjel). Detekce změny probíhá porovnáváním hodnoty masky měřeného úseku na aktuální pozici blobu a na jeho pozici v minulém snímku. Pokud došlo mezi snímky ke změně z 0 na 1, blob do úseku vjel, naopak změna z 1 na 0 značí, že úsek opustil.

  Pro zpřesnění této metody jsem v kapitole 4.2 popsal optimalizaci spočívající v rozlišování půlsnímků. Nejdříve je spočítána pozice blobu v půlsnímku, což představuje polovinu vzdálenosti mezi pozicí v aktuálním a předešlém snímku. Následně je odvozena pozice blobu ve čtvrtsnímcích, což je polovina vzdálenosti mezi pozicí v půlsnímku a v aktuálním snímku (respektive předchozím snímku). Dále se porovnává, jestli k vjetí do úseku (vyjetí z úseku) došlo už v prvním čtvrtsnímku (pak by se za hraniční označil předchozí snímek)

a pokud ne, tak se stejné porovnání provede se třetím čtvrtsnímkem (pak by byl hranici nejblíže půlsnímek). Jestliže ani tohle porovnání nebude pozitivní, tak je hraniční aktuální snímek, viz Obrázek 5.2.



Obrázek 5.2: Ukázka zpřesnění měření zavedením půlsnímků. Testuje se nejdříve 1. čtvrtsnímek a pokud leží uvnitř měřeného úseku, proběhla změna v předchozím snímku. Jinak se pokračuje testováním 3. čtvrsnímku a pokud se ten nachází uvnitř měřeného úseku, došlo ke změně v půlsnímku, jinak je změně nejbližší aktuální snímek. V konkrétně zobrazeném příkladě bude použit půlsnímek, který se nachází nejblíže hranici měřeného úseku.

5.3 Webová aplikace

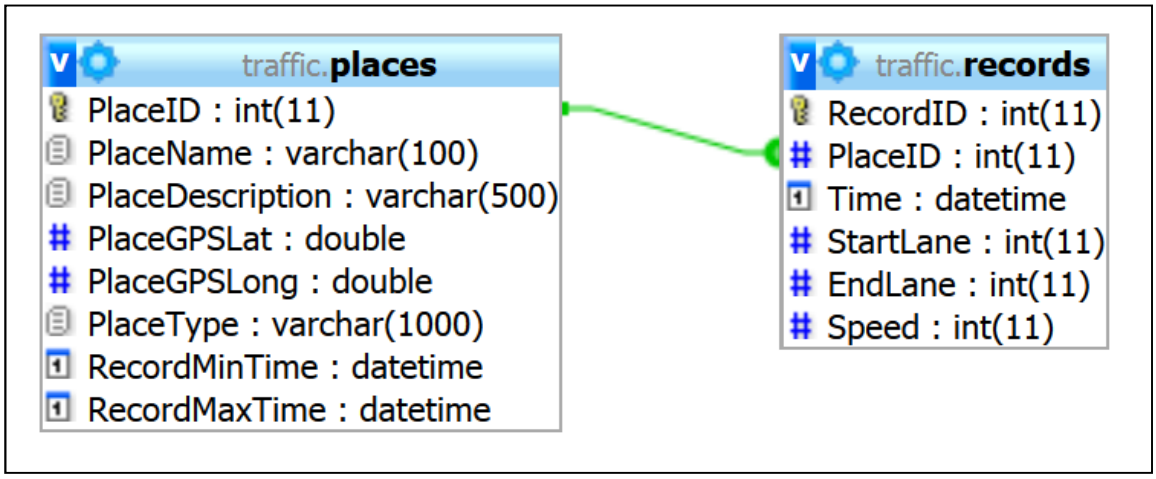
Aplikace pro zobrazení statistik je implementována jako webová aplikace s využitím technologie *Microsoft ASP.NET*, programovacího jazyka *C#*, javascriptové knihovny *jQuery* a služeb *Google Maps* a *Google Charts*. Vstupem aplikace jsou data z *MySQL* databáze, výstupem jsou statistiky za zvolené časové období (uživatel vybere počáteční a koncové datum) ve formě grafů a uživatelsky přívětivé rozhrání pro práci s nimi. Statistikami se konkrétně myslí denní statistiky, kdy se sleduje počet projetých vozidel a jejich průměrná rychlost pro každou hodinu. Implementace je rozdělena do následujících projektů:

- **`DataLayer`** – představuje nejnižší vrstvu aplikace, obsahuje třídy, na které se mapují data z databáze a pouze v tomto projektu je kód, který s databází přímo komunikuje,
- **`BussinessLayer`** – aplikační vrstva, obsahuje třídy představující statistiky vygenerované ze vstupních dat a veškerou logiku, podle které se statistiky vytváří,

- **`ScriptHelpers`** – podpůrný projekt, který obsahuje kód zajišťující správu javascriptů dynamicky generovaných do stránky,
- **`TrafficSurveillance`** – prezentační vrstva aplikace, obsahuje všechny webové soubory (stránky aspx, soubory s kaskádovými styly, javascriptové soubory, obrázky atd.) a s nimi spojenou logiku.

Databáze

Relační databáze představuje propojovací prvek mezi modulem pro sběr dat a webovou aplikací pro zobrazení statistik. Rozhodl jsem se použít databázový systém *MySQL*, protože je volně šiřitelný, rychlý, všeobecně známý a jednoduše nasaditelný v reálném provozu.



Obrázek 5.3: Schéma tabulek v databázi

Na Obrázku 5.3 je znázorněno schéma databáze. Je velmi jednoduché a sestává pouze ze dvou tabulek. V tabulce `Places` jsou uloženy informace o každé lokaci, neboli kameře, což představuje modul pro sběr dat. Primárním klíčem je identifikační číslo lokace, které je také uvedeno v konfiguračním souboru modulu pro sběr dat a je cizím klíčem v tabulce `Records`. Tímto je mezi záznamy v tabulkách `Places` a `Records` vytvořen vztah s kardinalitou 1:N.

Každý řádek v tabulce `Records` představuje jedno detekované vozidlo a obsahuje informace o čase detekce (sloupec `Time`), identifikátory pruhů, ve kterých se vozidlo pohybovalo na začátku (sloupce `StartLane`) a na konci sledování (sloupec `EndLane`), a odhad jeho rychlosti (sloupec `Speed`).

Tabulka `Places` o každé lokaci uchovává její jméno (`PlaceName`), popis (`PlaceDescription`), GPS souřadnice (`PlaceGPSLat` a `PlaceGPSLong`) a časy nejstaršího (`RecordMinTime`) a nejnovějšího (`RecordMaxTime`) záznamu v tabulce `Records`, který je dané lokaci přiřazen. Sloupec `PlaceType` nese informace o jednotlivých jízdních pruzích dané lokace ve formátu XML:

```
<lanelist xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <lane>
```

```
    <id>1</id>
    <title>levý pruh</title>
  </lane>
</lanelist>
```

Každý jízdní pruh představuje element `lane`. Ten obsahuje element `title` s názvem pruhu a element `id` s identifikátorem jízdního pruhu ($N >= 1$, odpovídá pořadí definicí jízdních pruhů v konfiguračním XML souboru modulu pro sběr dat).

V tabulce `Records` jsou vytvořeny indexy nad sloupci `PlaceID` a `Time`, protože se na základě nich velmi často vyhledávají a třídí záznamy. Dále tabulka `Records` obsahuje databázový trigger `record_inser_trigger`, který se spouští po každém vložení nového záznamu a zajišťuje automatickou aktualizaci polí `RecordMinTime` a `RecordMaxTime` v přiřazeném řádku tabulky `Places`.

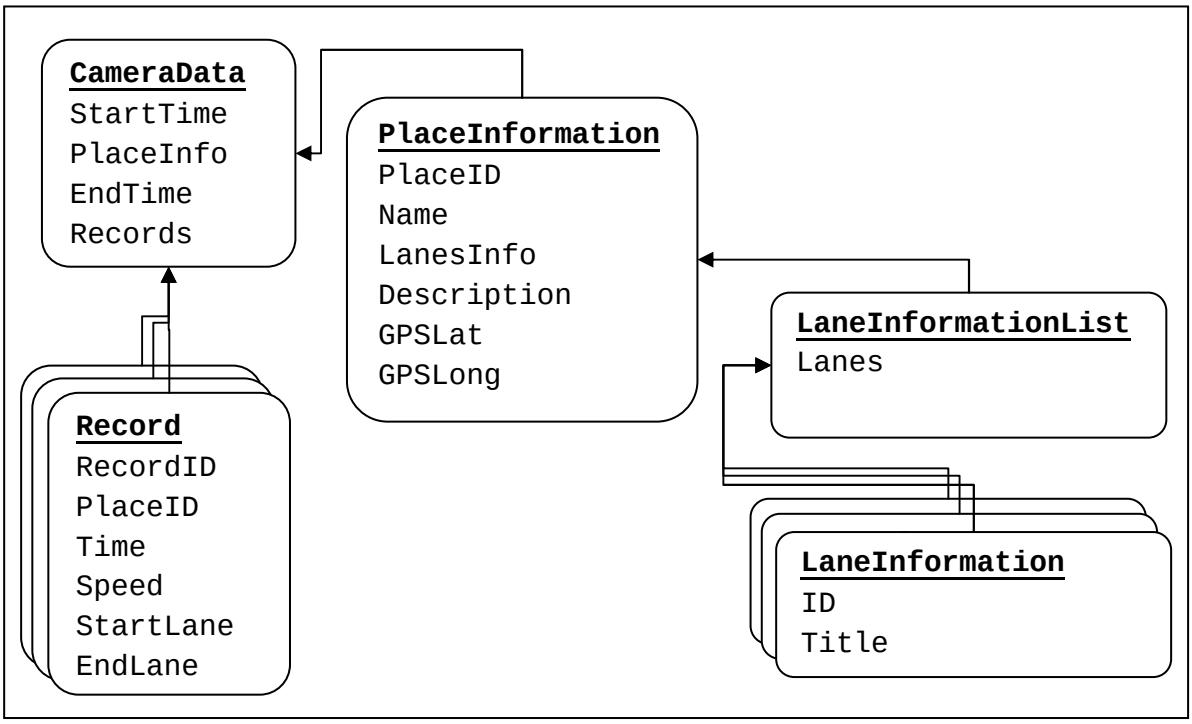
Datová vrstva

Datovou vrstvu představuje projekt `DataLayer`, který obsahuje tyto třídy:

- **`Record`** – Datová třída, ekvivalent záznamu o vozidle v databázi.
- **`PlaceInformation`** – Datová třída, ekvivalent záznamu o lokaci v databázi.
- **`LaneInformation`** a **`LaneInformationList`** – Datová třída, nese informace o jednotlivých jízdních pruzích lokace, které jsou v databázi ve formě XML záznamu v tabulce `Places`.
- **`CameraData`** – Datová třída, propojuje instance výše popsaných tříd do jednoho komplexnějšího celku, obsahuje všechny záznamy (třída `Record`) určité lokace (třída `PlaceInformation`) v daném časovém rozsahu, viz Obrázek 5.4.
- **`SessionHelper`** – Statická třída, používá se při ukládání objektů do paměti serveru.
- **`DatabaseHelper`** – Jediná třída, která přímo komunikuje s databází. Využívá externí konektor pro spojení s *MySQL* databází (*ADO.NET* ovladač od společnosti *Oracle*). Tato třída se připojí k databázi na základě *MySQL* připojovacího řetězce v souboru `web.config`:

  ```
  <connectionStrings>
        <add name="DatabaseConnection"
          connectionString="server=localhost;user id=root;
          password=root; database=traffic; pooling=false"/>
  </connectionStrings>
  ```

  Dále má veřejné metody určené pro komunikaci s databází (čtení i zápis).
- **`CameraHelper`** – Statická třída, sdružuje metody, které z databáze čtou data s využitím třídy `DatabaseHelper` a následně z nich vytváří objekty a ty spojují dohromady (například do třídy `CameraData`). Aplikační vrstva pro získání dat využívá právě tuto třídu.



Obrázek 5.4: Schéma propojení datových tříd

Aplikační vrstva

Aplikační vrstva je sdružena v projektu `BusinessLayer` a sestává z těchto tříd:

- **`Statistics`** – Datová třída, zapouzdřuje třídu datové vrstvy `CameraData`, přičemž má své vlastní vlastnosti `StartTime` a `EndTime`. S touto třídou se pracuje na úrovni aplikační vrstvy a slouží jako zdroj dat pro vytváření statistik. Třída `CameraData` má vlastnosti `StartTime` a `EndTime` určené podle načtených dat, kdežto třída `Statistics` má `StartTime` a `EndTime` nastavené podle časového úseku, za který chce uživatel vytvořit statistiky. Díky tomu je možné při změně časového úseku z databáze načítat pouze chybějící data, případně komunikaci s ní eliminovat úplně (pokud je nový časový úsek podmnožinou původního). Tato třída současně zařizuje uchovávání vstupních dat v paměti serveru (cachování), čímž se zvyšuje rychlost aplikace.
- **`StatisticsTraffic`** – Datová třída, nese informaci o průměrné rychlosti a počtu vozidel za časový úsek.
- **`StatisticsTrafficDayGraph`** – Datová třída, která se používá pro vytvoření denního grafu. Tvoří ji datum a dva seznamy instancí třídy `StatisticsTraffic`, kdy jeden seznam obsahuje agregovaná data pro každou denní hodinu a druhý seznam představuje agregovaná data za celý den pro jednotlivé jízdní pruhy.
- **`StatisticsHelper`** – Statická třída, sdružuje metody, které vytváří denní statistiky ze vstupních dat za využití komponenty *LINQ*.

- **`GlobalHelper`** – Pro aplikační vrstvu zapouzdřuje metody, které zprostředkují získání specifických dat z databáze přes datovou vrstvu, například informace o jednotlivých lokacích, seznam názvů všech lokací a podobně.

Prezentační vrstva

Projekt `TrafficSurveilance` implementuje uživatelské rozhraní webové aplikace a představuje nejvyšší vrstvu její architektury. Celé rozhraní sestává ze dvou *aspx* stránek, `Default.aspx` a `PlaceDetail.aspx`. Obě dodávají pouze obsah do hlavní stránky `MasterPage.Master`, která definuje jednotný vzhled celé aplikace.

- **`Default.aspx`** – Vstupní strana aplikace, zobrazí všechny lokace v mapě (služba *Google Maps*) podle jejich GPS souřadnic a přímo u nich barevně znázorní aktuální dopravní situaci. Zelená barva znamená, že je provoz plynulý, červená barva značí dopravní problém. Informace o aktuální situaci jsou získána na základě průměrných rychlostí vozidel, která daným úsekem projela za posledních 15 minut. Barva aktuální situace je aproximována dynamicky v javascriptu, kdy dochází k míchání červené a zelené barvy v závislosti na tom, jak moc se aktuální průměrná rychlost blíží nastaveným hraničním hodnotám. Tím je dosaženo plynulého, nikoliv skokového, přechodu mezi zelenou a červenou barvou v případě změny dopravního stavu.

  Po kliknutí na ikonu dopravní kamery v mapě je uživatel přesměrován na detailní zobrazení konkrétního úseku.

- **`PlaceDetail.aspx`** – Tato stránka zobrazí všechny dostupné informace o zvolené lokaci včetně statistik. Je zjištěno datum a čas nejstaršího a nejnovějšího záznamu v databázi pro danou lokaci a jsou vytvořeny denní statistiky za celý poslední den. Všechna data, která jsou získána z aplikační vrstvy, se následně serializují do javascriptových objektů. O vykreslení údajů a grafů se následně postarají javascriptové funkce, které jsou postupně volány ihned po kompletním vytvoření objektového modelu dokumentu výsledné HTML stránky. Javascriptové funkce za použití knihovny *jQuery* a *Google API* zajistí následující:

  - zobrazí satelitní pohled na danou lokaci ve službě *Google Maps*,
  - vytvoří vysouvací menu, které umožňuje okamžitě přejít na jinou lokaci,
  - vytvoří kalendář pro výběr časového okna, pro které chce uživatel zobrazit statistiky (knihovna *jQuery UI*),
  - pokud jsou zobrazeny statistiky za více dní, zajistí vytvoření ovládacího prvku pro přechod mezi zobrazením jednotlivých denních statistik (plugin *jQuery Timelinr*),
  - zobrazí grafy pro aktuálně zvolený den (služba *Google Chart Tools*).

  Ovládání stránky je poté řízeno pouze javascriptem s využitím technologie *AJAX*. Pokud uživatel zvolí jiné časové období, pro které chce zobrazit statistiky, provede se asynchronní požadavek na server. Vrácená data jsou použita k překreslení pouze části stránky obsahující grafy, neprovádí se překreslení celé stránky.

6 Výsledky a testování

V této kapitole budou popsány výsledné aplikace, které vznikly v rámci bakalářské práce. Dále bude nastíněno, jakým způsobem se s nimi pracuje, a nakonec budou uvedeny výsledky testování, kterému byly aplikace podrobeny a které ověřovalo jejich funkčnost.

6.1 Modul pro sběr dat

Program představující modul pro sběr dat se jmenuje CarTracker a je spouštěn následujícím příkazem:

```
    CarTracker.exe [-v] [-i infile.avi] [-o outfile.xml]
[-r outfile.avi] [-c config.xml] [-t "2012-03-01 15:30"]
```

Vstupem programu je videosignál z připojené kamery nebo videosoubor ve formátu avi, pokud je jeho jméno zadáno volitelným parametrem `-i`. Konfigurace programu se standardně načítá ze souboru *config.xml*, který se nachází ve stejné složce jako spustitelný soubor *CarTracker.exe*, nebo volitelně ze souboru zadaného přes paramter `-c`.

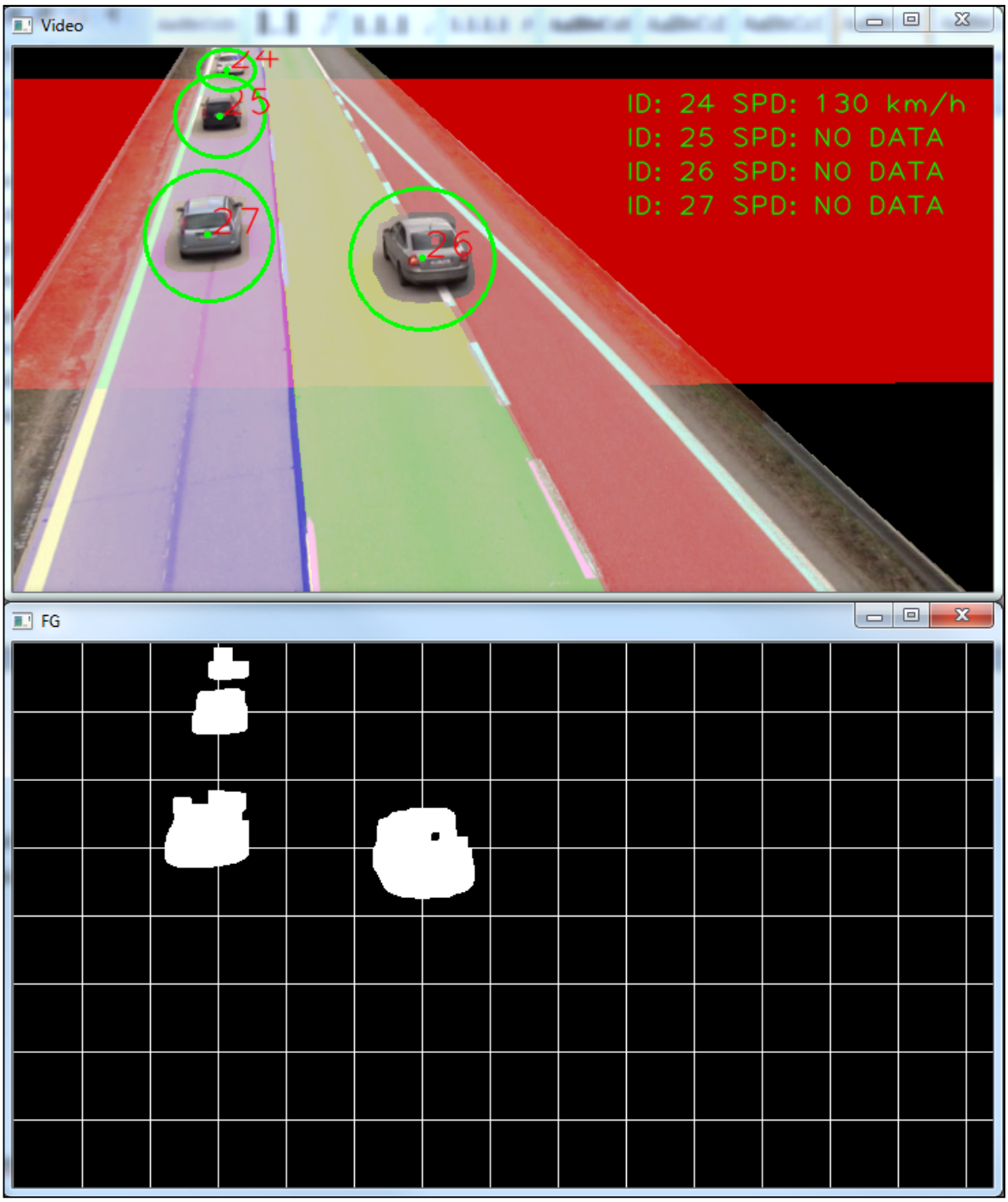
Výstup modulu pro sběr dat je zapisován do MySQL databáze, ke které se program připojí na základě nastavení v konfiguračním souboru. Dále může být výstupem i XML soubor specifikovaný volitelným parametrem `-o`. Tento soubor může sloužit jako záložní řešení pro případ výpadku připojení k databázi nebo může být i jediným výstupem, jestliže v konfiguračním souboru chybí sekce týkající se nastavení databáze.

Po spuštění programu se okamžitě začne zpracovávat vstupní videosignál a za čas detekce jednotlivých vozidel je považován aktuálně nastavený systémový čas. V případě zpracování dříve zaznamenaného videosouboru je možné nastavit počáteční datum a čas parametrem `-t` ve formátu `"RRRR-MM-DD HH-MM"`, který začne plynout od prvního snímku. Další parametry programu jsou popsány v *Manuálu*, který je na přiloženém DVD.

Po spuštění programu je možné používat klávesové zkratky pro různé funkce, které uživateli mohou pomoci se správným nastavením modulu pro konkrétní lokaci. Například je možné zobrazit:

- okno s aktuálním popředím scény po všech optimalizacích,
- do původní videosekvence vykreslit:
    - nadefinovanou masku oblasti zájmu,
    - oblast používanou pro měření rychlosti,
    - jednotlivé nadefinované jízdní pruhy, které jsou automaticky barevně odlišeny,
    - vypisovat přehled odhadnutých rychlostí vozidel, které jsou momentálně v záběru.

Ukázka uživatelského rozhraní aplikace je na Obrázku 6.1. Veškeré funkce a jim přiřazené klávesové zkratky jsou popsány v *Manuálu*.



Obrázek 6.1: Ukázka uživatelského rozhraní aplikace. Nahoře okno s aktuálně zpracovávaným videosignálem. V něm je zobrazena pouze nadefinovaná oblast zájmu z původního videosignálu, dále je červeně vyznačena oblast měření rychlosti vozidel, barevně jsou odlišeny jednotlivé jízdní pruhy a kolem sledovaných vozidel je vykreslena elipsa s pořadovým číslem vozidla. Vpravo jsou vypsány rychlosti aktuálně sledovaných vozidel, nicméně pouze vozidlo číslo 24 již opustilo měřený úsek a má tedy dostupná data o své rychlosti. Dole okno s aktuálním popředím scény po optimalizacích.

Konfigurace programu

Konfiguračním XML souborem je možné ovlivnit nastavení programu pro konkrétní lokaci, například (přesný zápis všech nastavení je vysvětlen v *Manuálu*):

- připojení k databázi,
- identifikátor kamery (více viz kapitola 5.3),
- nastavení modelu pozadí a klasifikace blobů,
- definice oblastí, ty jsou určeny seznamem bodů v obraze (x a y souřadnice v pixelech, počátek souřadnic v levém horním rohu obrazu) a tvoří vrcholy monotónního polygonu:
    - oblast zájmu ve scéně,
    - oblast úseku pro měření rychlosti a její délka v metrech,
    - oblasti určující jednotlivé jízdní pruhy.

Testování programu

Program byl průběžně testován na několika videosekvencích, které jsem natočil z mostu nad dálnicí D1 *FullHD* kamerou (rozlišení 1920×1080 pixelů, 25 *FPS*) za použití stativu. Původní záznamy byly převzorkovány na rozlišení 720×400 pixelů. K finálnímu testování jsem vybral videosekvence *D1_182_Brno.avi*, která zabírá menší část vozovky (cca 100 metrů), a *D1_182_Praha.avi*, která má v záběru cca 500 metrů vozovky, viz Obrázek 6.2. Oba testovací soubory jsou dostupné na přiloženém DVD včetně použité konfigurace programu.

a) D1_182_Brno.avi *b) D1_182_Praha.avi*

Obrázek 6.2: Ukázky testovacích videosekvencí

Z testovacích videosekvencí byla ručně získána referenční data (bylo zaznamenáno každé vozidlo, které se objeví v oblasti zájmu scény a byl mu určen jízdní pruh v době vstupu do záběru). Tato referenční data byla poté porovnána s výstupy programu v databázi, výsledek porovnání je v tabulkách 6.1 a 6.2, přičemž význam sloupců je následující:

- **ID** – referenční identifikátor vozidla (pořadí vozidla ve videosekvenci),
- **T** – typ vozidla (O = osobní, N = nákladní, M = motocykl),
- **JP** – referenční identifikátor jízdního pruhu, ve kterém se vozidlo pohybovalo,
- **Čas** – čas detekce vozidla vůči začátku videosekvence (X = vozidlo nebylo detekováno),
- **DP** – identifikátor programem detekovaného jízdního pruhu (X = nedostupná data),
- **V** – programem vypočítaný odhad rychlosti vozidla (X = nedostupná data).

ID	T	JP	Čas	DP	V		ID	T	JP	Čas	DP	V		ID	T	JP	Čas	DP	V
1	O	2	0:08	2	116		22	O	1	0:46	1	118		43	O	1	1:51	1	123
2	O	1	0:10	1	128		23	O	2	0:46	2	89		44	O	1	1:53	1	134
3	O	1	0:11	1	64		24	O	1	0:59	1	142		45	N	2	1:56	2	X
4	N	3	0:12	3	X		25	O	3	1:00	3	X		46	O	1	1:58	1	138
5	O	1	0:14	1	116		26	O	1	1:03	1	132		47	N	2	2:03	2	X
6	O	2	0:15	2	116		27	O	1	1:06	1	132		48	O	1	2:06	1	116
7	O	1	0:17	1	128		28	O	1	1:07	1	123		49	N	2	2:07	2	X
8	O	1	0:20	1	128		29	O	1	1:08	1	130		50	O	1	2:08	1	107
9	O	1	0:23	1	115		30	O	1	1:08	1	147		51	O	1	2:09	1	120
10	N	2	X	X	X		31	O	3	1:08	3	81		52	O	3	2:11	3	128
11	O	1	0:25	1	X		32	O	2	1:13	2	125		53	O	2	2:14	2	54
12	O	1	0:25	1	120		33	O	2	1:18	2	130		54	O	1	2:17	1	123
13	O	3	0:26	3	95		34	O	3	1:23	3	93		55	O	1	2:17	1	105
14	O	1	0:27	1	116		35	O	2	1:27	2	121		56	O	3	2:20	3	115
15	O	1	0:29	1	123		36	O	2	1:33	2	118		57	O	1	X	X	X
16	O	1	0:31	1	121		37	O	2	1:35	2	121		58	N	2	X	X	X
17	O	3	0:31	3	X		38	O	1	1:42	1	44		59	N	2	X	X	X
18	O	3	0:36	3	97		39	O	2	1:42	2	46		60	O	2	2:27	2	100
19	O	3	0:40	3	97		40	O	2	1:42	2	38		61	N	2	2:28	2	X
20	O	1	0:41	1	111		41	N	2	X	X	X		62	O	1	2:29	1	111
21	O	2	0:43	2	92		42	O	1	1:46	1	X		63	O	1	2:30	1	111

Tabulka 6.1: Test programu na videosekvenci D1_182_Brno.avi.

ID	T	JP	Čas	DP	V		ID	T	JP	Čas	DP	V		ID	T	JP	Čas	DP	V
1	O	2	0:00	2	97		21	N	2	0:54	2	93		41	O	2	1:31	2	120
2	O	3	0:00	3	109		22	O	3	0:56	3	97		42	O	2	1:33	2	130
3	O	1	0:01	1	135		23	O	1	0:56	1	139		43	O	1	1:35	1	162
4	O	1	0:03	1	126		24	O	1	0:59	1	150		44	O	2	1:37	2	128
5	O	2	0:09	2	135		25	O	3	1:02	3	119		45	O	1	1:38	1	155
6	O	2	0:21	2	142		26	O	1	1:03	1	132		46	O	2	1:40	2	142
7	N	3	0:22	3	92		27	O	2	1:04	2	114		47	O	3	1:42	3	124
8	M	3	0:23	3	X		28	O	1	1:13	1	158		48	O	1	1:44	1	X
9	O	3	0:24	3	75		29	O	2	1:13	2	114		49	O	3	1:51	3	109
10	O	3	0:25	3	84		30	O	1	1:14	1	147		50	N	2	1:55	2	100
11	O	3	0:26	3	81		31	O	2	1:14	2	137		51	O	3	1:57	3	120
12	O	2	0:32	2	120		32	O	3	1:14	3	135		52	O	2	2:01	2	76
13	O	3	0:36	3	101		33	O	2	1:21	2	126		53	O	2	2:01	2	109
14	O	1	0:37	1	109		34	N	2	1:24	2	114		54	O	2	2:06	2	139
15	O	2	0:37	2	147		35	O	1	1:26	1	135		55	O	2	2:08	2	85
16	O	3	0:40	3	96		36	O	1	1:28	1	128		56	N	3	2:11	3	96
17	O	2	0:41	2	139		37	N	3	1:28	3	117		57	O	2	2:12	2	120
18	O	2	0:45	2	110		38	O	2	1:29	2	X		58	O	2	2:14	2	122
19	O	1	0:48	1	162		39	O	1	1:29	1	126		59	O	3	2:20	3	128
20	N	3	0:54	3	X		40	O	3	1:30	3	108							

Tabulka 6.2: Test programu na videosekvenci D1_182_Praha.avi

Pro vyšší přehlednost jsou údaje v tabulkách barevně označeny. Zelené pozadí značí shodu výstupu programu s referenčním údajem, oranžové pozadí značí nesprávné nebo chybějící výstupy, červené pozadí znamená, že vozidlo nebylo programem v záběru vůbec detekováno.

Vzhledem k tomu, že cílem aplikace jsou souhrnné statistiky, bylo testování zaměřeno na přítomnost relevantních dat v databázi, nikoliv na přesné sledování trajektorie jednotlivých vozidel v záběru. Vozidlo je úspěšně detekováno, pokud o něm v databázi existuje záznam (čas detekce v databázi musí ležet v časovém intervalu určeném přítomností vozidla v záběru).

Jestliže záznam existuje, je zjišťováno, zda byl správně určen jízdní pruh. Dále se sleduje dostupnost údaje o rychlosti vozidla (ten je vyhodnocen pouze v případě, kdy blob představující vozidlo byl detekován před měřeným úsekem a následně jej celý projel) a jestli při jeho měření nedošlo k evidentnímu zkreslení výsledné hodnoty (došlo ke spojení blobů pomalejšího a rychlejšího vozidla v měřeném úseku apod.). Přesnost odhadu rychlosti vozidla nelze určit, protože nejsou dostupná žádná referenční data.

Pokud o vozidle neexistuje záznam v databázi, je vozidlo považováno za nedetekované. Jestliže se v databázi vyskytuje záznam, který nelze přiřadit k referenčním datům, jedná se o falešnou detekci (například v důsledku opětovné detekce vozidla vyřazeného ze sledování, segmentace blobu vozidla, detekce pohybu stínů mraků na vozovce apod.).

Z výsledků v tabulce 6.1 (krátký záběr) vyplývá následující:

- úspěšnost detekce vozidel byla 92% (58 vozidel detekováno z 63 ve videosekvenci),
- nedošlo k žádné falešné detekci,
- pokud bylo vozidlo programem detekováno, tak:
    - úspěšnost detekce jízdního pruhu byla 100% (58 vozidlům z 58 detekovaných byl přiřazen shodný jízdní pruh jako v referenčních datech),
    - v 84,5% případů byla odhadnuta rychlost vozidla (49 vozidel z 58 detekovaných mělo dostupný odhad rychlosti),
    - v případě existujícího odhadu rychlosti je údaj relevantní v 89,8% případů (5 odhadů z dostupných 49 bylo označeno za zkreslené),
    - celková úspěšnost odhadu rychlosti po vyloučení zkreslených odhadů je tedy 75,9% (44 odhadů z 58 detekovaných vozidel).

Shrnutí výsledků z tabulky 6.2 (dlouhý záběr) je následovné:

- úspěšnost detekce vozidel byla 100% (detekováno 59 z 59 vozidel),
- nedošlo k žádné falešné detekci, přestože se ve scéně často mění světelné podmínky (natočeno za slunečného počasí, po vozovce se pohybují stíny mraků),
- úspěšnost detekce jízdního pruhu byla 100% (59 vozidel z 59 detekovaných mělo přiřazený stejný jízdní pruh, jako byl referenční),
- v 93,2% případů byla odhadnuta rychlost vozidla (55 vozidel z 59 detekovaných mělo dostupný odhad rychlosti).

Z výše uvedeného vyplývá, že lepších výsledků bylo dosaženo v případě videosekvence *D1_182_Praha.avi*. Je tedy lepší kameru umístit tak, aby zabírala delší úsek vozovky. Příčina zhoršené detekční schopnosti ve videosekvenci *D1_182_Brno.avi* je právě v krátkém záběru, kdy nákladní automobily zabírají velkou část scény. Díky tomu může dojít i k automatické korekci

expozice kamery v situaci, kdy do scény vstoupí nákladní automobil s kontrastní bílou plachtou návěsu. Blob detektor navíc hledá pouze ty spojité části popředí, které se nedotýkají hrany obrazu, což představuje problém právě u dlouhých nákladních automobilů s přívěsem.

Dalším problémem krátkého záběru je zhoršená schopnost vypočítání odhadu rychlosti vozidla, protože měřený úsek zabírá příliš velkou část scény a některá vozidla jsou detekována, až když se jejich referenční bod nachází uvnitř tohoto úseku (v momentě kdy se jim náležící blob oddělí od hrany obrazu). Na druhou stranu ani delší záběr by nedokázal vyřešit problém, který nastal v čase 2:25, kdy do záběru vstupují současně tři překrývající se vozidla a velký blob vzniklý jejich spojením nemohl být kvůli svým rozměrům považován za samostatné vozidlo, viz Obrázek 6.2 a). Bylo by zajímavé ještě experimentovat s jiným nastavením kamery při pořizování záznamu, pro detekci každého vozidla by například mohla dobré výsledky podávat kamera s objektivem typu „rybí oko“ zavěšená pod vyšším mostem a namířená kolmo na vozovku. Při velkém zkreslení by ale na druhou stranu bylo složitější odhadovat rychlosti vozidel.

K výsledkům testu videosekvence *D1_182_Praha.avi* bych chtěl dodat, že se jedná o scénu poměrně náročnou na vhodné nastavení parametrů trackeru a modelu pozadí, protože je natočena za slunečného dne, kdy mraky vrhají ostré pohybující se stíny na vozovku. Aby nebyl detekován pohyb těchto stínů, bylo nutné snížit citlivost detekce popředí, což ale současně přirozeně vedlo ke zhoršení detekce vozidel. Tracker v této scéně díky tomu musí sledovat i poměrně malé bloby, což je navíc umocněno perspektivou dlouhého záběru.

Pokud se výsledky obou videosekvencí vyhodnotí souhrnně, je celková úspěšnost detekce vozidel 95,9% (117 vozidel detekováno ze 122) a úspěšnost určení správného jízdního pruhu je v případě úspěšné detekce 100% (117 vozidel ze 117). Jestliže se na výsledky podíváme z pohledu cíle aplikace, která má uživateli nabídnout především souhrnné statistiky, lze ji označit za reálně použitelnou.

6.2 Webová aplikace

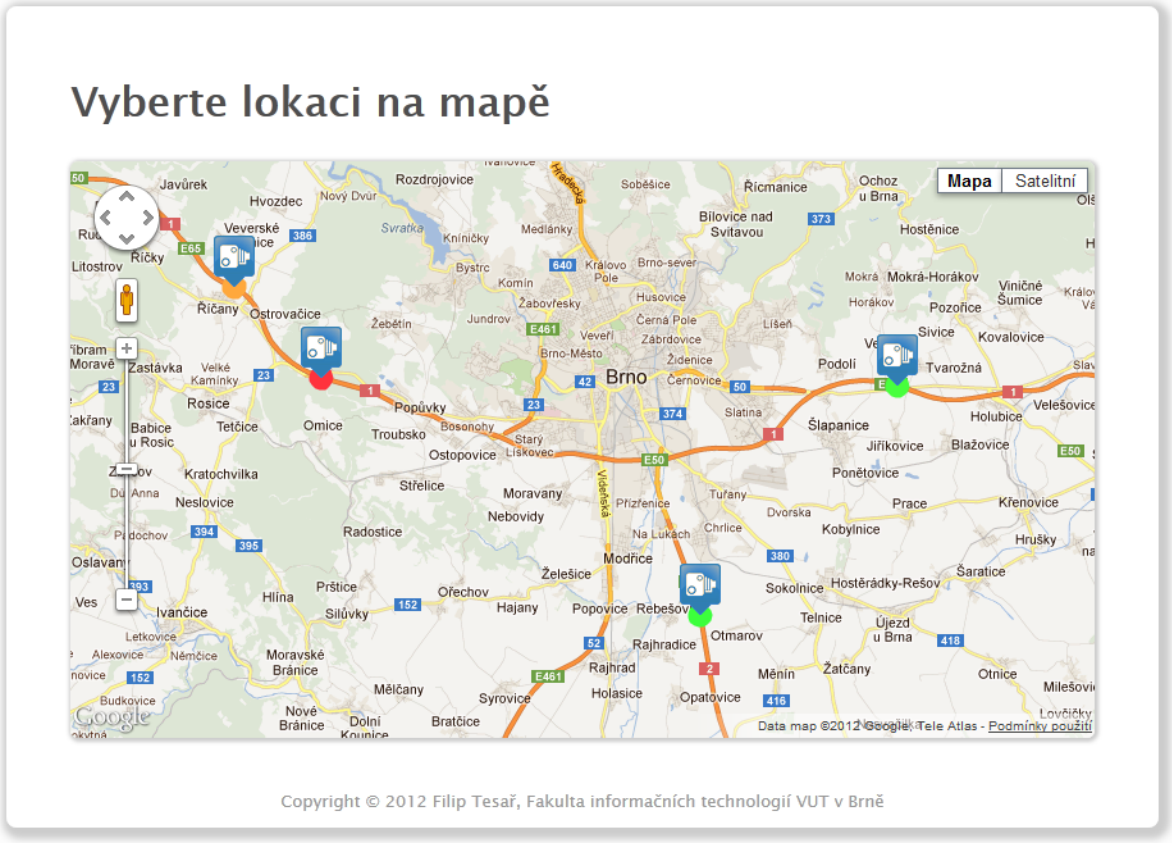
Aplikace pro zobrazení statistik umožňuje uživateli zobrazit mapu s přehledem aktuálních dopravních stavů všech lokací, viz Obrázek 6.3. Na této mapě je možné klikem na ikonu kamery vybrat konkrétní lokaci, pro kterou chce uživatel zobrazit podrobné statistiky.

Na stránce s podrobnými statistikami (Obrázek 6.5) jsou zobrazeny informace o lokaci, satelitní pohled na daný úsek vozovky a následující grafy vztažené k vybranému dni:

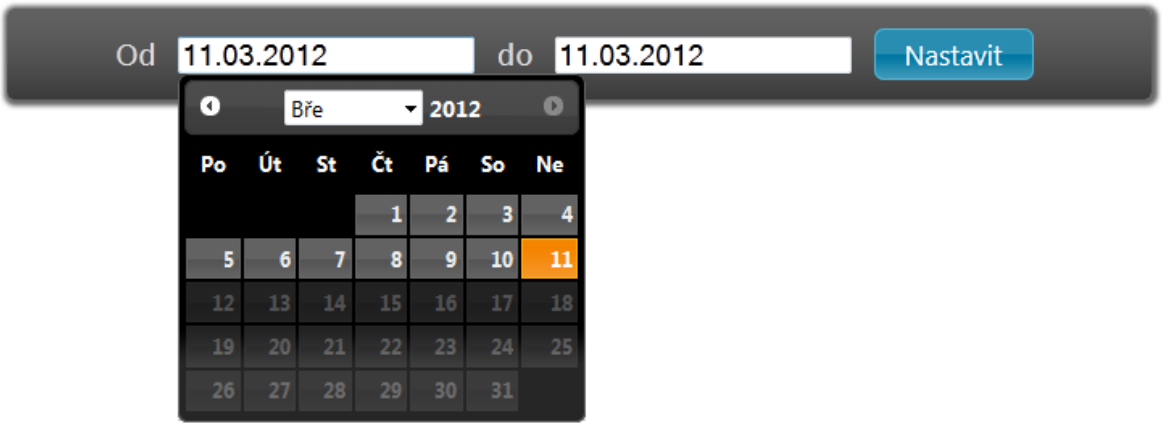
- celkový počet projetých vozidel za každou hodinu dne,
- průměrná rychlost těchto vozidel (za každou hodinu),
- celkový počet vozidel v jednotlivých jízdních pruzích za celý den,
- průměrná rychlost vozidel v jednotlivých jízdních pruzích za celý den.

Po otevření stránky jsou nejdříve zobrazeny statistiky vztahující se k poslednímu možnému dni vzhledem k dostupnosti dat v databázi. Uživatel si poté může vybrat jiný časový úsek pomocí dvou kalendářů (počáteční datum a koncové datum, viz Obrázek 6.4), přičemž nemůže zvolit rozpětí, pro které nejsou dostupná data v databázi. Pokud je vybráno více dnů, tak je možné mezi jejich zobrazením přepínat ovládacím prvkem pod kalendářem (kliknutím na datum jiného vybraného dne).

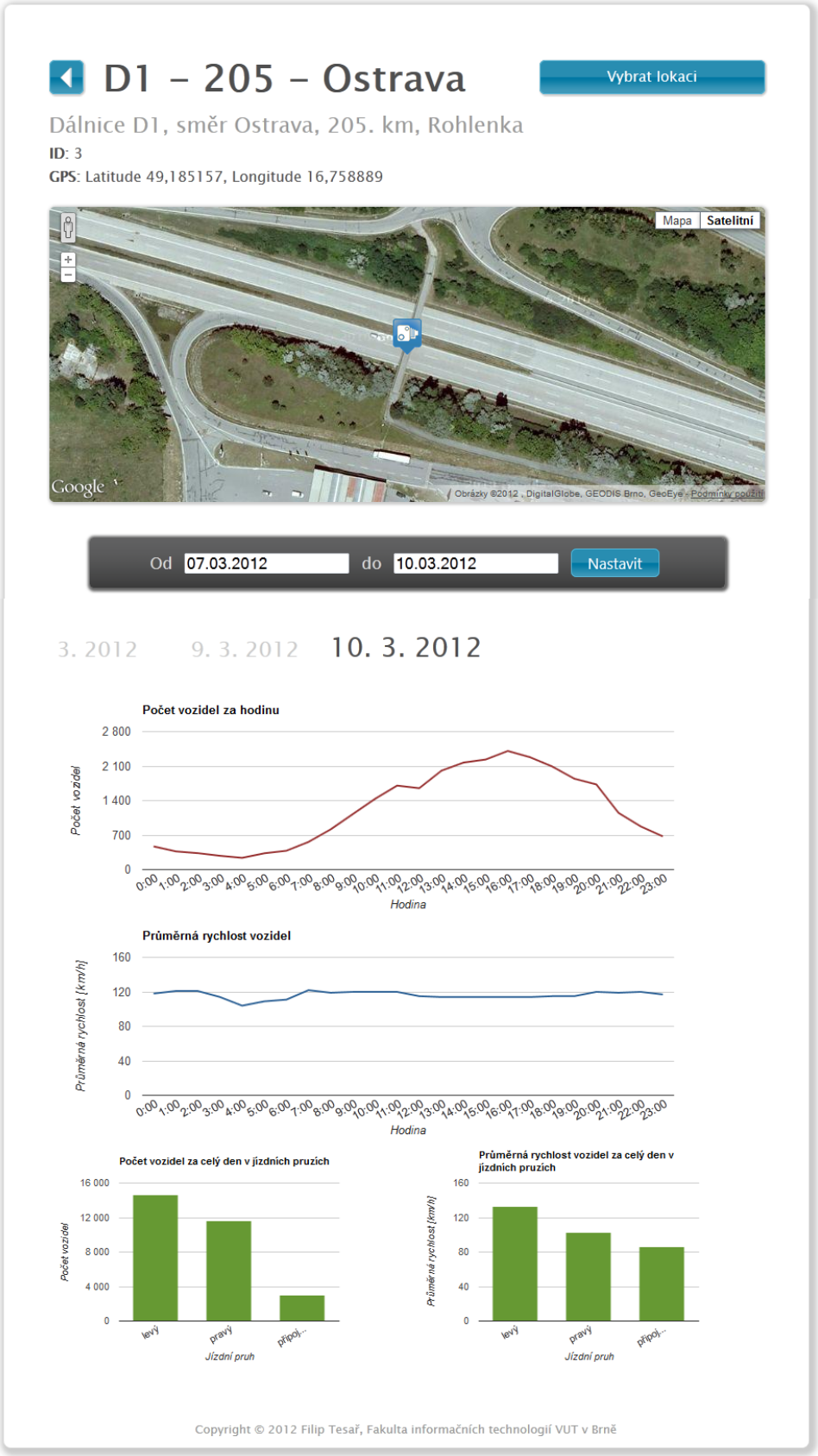
Testování aplikace bylo provedeno na výstupech z modulu pro sběr dat a navíc byla použita vygenerovaná data simulující několikatýdenní sledování čtyř lokací. Na těchto simulovaných datech byla poté aplikace odladěna a optimalizována.



Obrázek 6.3: Ukázka výběru lokace na mapě s barevným odlišením aktuálních dopravních stavů



Obrázek 6.4: Ukázka kalendáře pro výběr časového období



Obrázek 6.5: Celkový výstup aplikace pro konkrétní lokaci

7 Závěr

Cílem práce byl návrh a implementace aplikace, která z videosekvence zachycující dopravní provoz bude sbírat statistická data o dopravě na sledovaném úseku komunikace a z těchto dat bude následně vytvářet grafy pro analýzu dopravní situace. Jednou z funkcí aplikace mělo být také zjednodušené zobrazení aktuálních dopravních stavů z více kamer na jedné obrazovce.

Výsledná aplikace byla rozdělena na dvě samostatné části – modul sbírající data implementovaný v jazyce C++ a webovou aplikaci implementovanou v jazyce C# s využitím platformy ASP.NET. Aplikace byly mezi sebou propojeny relační databází.

Vytvořená aplikace umožňuje sledování více dopravních úseků současně nezávislými kamerami, přičemž o každém úseku jsou sbírána statistická data, například počet projíždějících vozidel, je odhadována jejich rychlost a detekován jízdní pruh, ve kterém se pohybují. Z těchto dat jsou následně webovou aplikací vytvářeny grafy pro analýzu dopravní situace. Webová aplikace také dokáže v mapě zobrazit aktuální stupeň provozu všech sledovaných úseků současně.

Z testování aplikace vyplynulo, že by výsledný systém mohl fungovat v reálném provozu pro vytváření souhrnných statistik o počtu projíždějících vozidel a vytíženosti jednotlivých jízdních pruhů. Předpokladem je vhodně zvolené umístění kamery, úhel jejího záběru a konkrétní scéně přizpůsobená konfigurace detektoru. Vhodnost využití aplikace pro měření rychlostí vozidel nebyla určena vzhledem k nedostupnosti referenčních dat k porovnání.

Budoucí rozšíření práce by mohlo spočívat v použití pokročilejší metody sledování objektů, například s využitím predikce Kalmanova filtru. Dále by bylo zajímavé experimentovat s použitím kamery s objektivem typu "rybí oko", který by při záběru kolmo k vozovce mohl eliminovat problém překrývajících se vozidel. Dalším vhodným vylepšením by byla větší integrace modulu pro sběr dat a webové aplikace pro zobrazení statistik. Například konfigurace detektoru by mohla být prováděna vzdáleně přes webové rozhraní. S tím souvisí i možnost přenosu reálného obrazu z kamery přímo do webové aplikace.

Literatura

1. **Cheung, S.-C. S. a Kamath, C.** Robust techniques for background subtraction in urban traffic video. *EURASIP Journal on Applied Signal Processing.* New York, United States : Hindawi Publishing Corp., 1. 1 2005. Sv. 2005, stránky 2330-2340. ISSN: 1110-8657 doi:10.1155/ASP.2005.2330.

2. **Collins, R.** Lecture 24 Video Change Detection. *CSE/EE486 Computer Vision I.* [Online] 2007. http://www.cse.psu.edu/~rcollins/CSE486/lecture24.pdf.

3. **Piccardi, M.** Background subtraction techniques: a review. *Systems, Man and Cybernetics, 2004 IEEE International Conference on.* 10-13. 10 2004. Sv. 4, stránky 3099-3104. doi: 10.1109/ICSMC.2004.1400815.

4. **Stauffer, C. a Grimson, W.E.L.** Adaptive background mixture models for real-time tracking. *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.* Cambridge : IEEE Computer Society Conference on Computer Vision and Pattern, 1999. Sv. 2. doi: 10.1109/CVPR.1999.784637.

5. **Intille, S. S., Davis, J. W. a Bobick, A. F.** Real-Time Closed-World Tracking. *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on.* Cambridge : autor neznámý, 17-19. 6 1997. stránky 697-703. doi: 10.1109/CVPR.1997.609402.

6. **Collins, R.** Lecture 28 Intro to tracking. *CSE/EE486 Computer Vision I.* [Online] 2007. http://www.cse.psu.edu/~rcollins/CSE486/lecture28.pdf.

7. **Narayana, M.** Automatic Tracking of Moving Objects in Video for Surveillance Applications. Lawrence : The University of Kansas, 2007.

8. **Yang, T., a další.** Real-time Multiple Objects Tracking with Occlusion Handling in Dynamic Scenes. *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on.* 20-25. 6 2005. Sv. 1, stránky 970-975. doi: 10.1109/CVPR.2005.292.

9. **Chen, Z.** *Bayesian Filtering: From Kalman Filters to Particle Filters, and Beyond.* 2003.

10. **Silniční vývoj - ZDZ spol. s r.o.** *TP 58 SMĚROVÉ SLOUPKY A ODRAZKY. ZÁSADY PRO POUŽÍVÁNÍ.* Brno : Ministerstvo dopravy, Odbor pozemních komunikací, 2008.

11. **Seidl, A.** *TP 133 Zásady pro vodorovné dopravní značení na pozemních komunikacích (II. vydání), Aktualizace 2011.* Praha : Ministerstvo dopravy, Odbor pozemních komunikací, 2011.

12. **Willow Garage.** Welcome - OpenCVWiki. *OpenCVWiki.* [Online] 18. 4 2012. [Citace: 24. 4 2012.] http://opencv.willowgarage.com/wiki/.

13. **Microsoft.** .NET Development. *MSDN - Explore Windows, Web, Cloud, and Windows Phone Software Development.* [Online] 2012. [Citace: 25. 4 2012.] http://msdn.microsoft.com/en-us/library/ff361664%28v=vs.110%29.aspx.

14. —. LINQ (Language-Integrated Query). *MSDN – Explore Windows, Web, Cloud, and Windows Phone Software Development.* [Online] 2012. [Citace: 25. 4 2012.] http://msdn.microsoft.com/en-us/library/bb397926.aspx.

15. **Wikipedia.** MySQL - Wikipedia, the free encyklopedia. *Wikipedia, the free encyklopedia.* [Online] 23. 4 2012. [Citace: 24. 4 2012.] http://en.wikipedia.org/wiki/MySQL.

16. **The jQuery Foundation.** jQuery: The Write Less, Do More, JavaScript Library. *jQuery JavaScript Library.* [Online] 2012. [Citace: 24. 4 2012.] http://jquery.com/.

17. **KaewTraKulPong, P. a Bowden, R.** *An Improved Adaptive Background Mixture Model for Realtime Tracking with Shadow Detection.* Middlesex : Brunel University, 2001.

18. **Chen, T. P., a další.** *Computer Vision Workload Analysis: Case Study of Video Surveillance Systems.* místo neznámé : Intel Technology Journal, 2005. ISSN 1535-864X.
19. **Bradski, G. a Kaehler, A.** *Learning OpenCV.* Sebastopol : O'Reilly Media, Inc., 2008. ISBN: 978-0-596-51613-0.

Seznam příloh

Příloha 1. DVD obsahující:

- text této bakalářské práce ve formátu PDF,
- zdrojový soubor textu této bakalářské práce,
- zdrojové texty implementovaných aplikací,
- spustitelné soubory implementovaných aplikací,
- zdrojové texty a DLL soubory použitých knihoven,
- uživatelský manuál,
- plakát prezentující tuto práci.